А. В. БАТАЕВ, Н.Ю.НАЛЮТИН,
С. В. СИНИЦЫН

ОПЕРАЦИЯЛЫҚ ЖҮЙЕЛЕР ЖӘНЕ ОРТАЛАР

Оқулық

«Федералдық білім беруді дамыту институты» федералды мемлекеттік автономды мекемесімен («ФБДИ» ФМАМ) бастапқы кәсіптік білім беру бағдарламасын іске асыратын білім беру мекемелерінің, 230000 «Ақпараттандыру және есептеуіш техникалары» мамандығы тобын оқыту процесінде оқу құралы ретінде қолдануға ұсынылады.

Рецензиялаудың тіркелген нөмірі 559
2013 жылғы 20 желтоқсан «БДФИ» ФМАМ

Мәскеу
«Академия» баспасөз орталығы
2014

ӘОЖ 681.3.066(075.32)
КБЖ 32.973-018.2я723
Б28

Бұл кітап Қазақстан Республикасының Білім және ғылым министрлігі және «Кәсіпқор» холдингі» КЕАҚ арасында жасалған шартқа сәйкес ««ТжКБ жүйесі үшін шетел әдебиетін сатып алуды және аударуды ұйымдастыру жөніндегі қызметтер» мемлекеттік тапсырмасын орындау аясында қазақ тіліне аударылды. Аталған кітаптың орыс тіліндегі нұсқасы Ресей Федерациясының білім беру үдерісіне қойылатын талаптардың ескерілуімен жасалды.

Қазақстан Республикасының техникалық және кәсіптік білім беру жүйесіндегі білім беру ұйымдарының осы жағдайды ескеруі және оқу үдерісінде мазмұнды бөлімді (технология, материалдар және қажетті ақпарат) қолдануы қажет.

Аударманы «Delta Consulting Group» ЖШС жүзеге асырды, заңды мекенжайы: Астана қ., Иманов көш., 19, «Алма-Ата» БО, 809С , телефоны: 8 (7172) 78 79 29, эл. поштасы: info@dcg.kz

П і к і р б е р у ш і :

Мәскеу мемлекеттік радиотехника, электроника және автоматика техникалық университеті кибернетика факультетінің деканы, техника ғылымдарының докторы, проф. *М. П. Романов;*

Мәскеу электроника және математика НИУ ВШЭ институты, кибернетика кафедрасының профессоры, техн. ғылымдары д-р *Б. И. Прокопов*

Батаев А.В.

Б28 Операциялық жүйелер және орталар: орта кәсіптік білім беру мекемелерінің студенттеріне арналған оқу құралы / А. В. Батаев, Н. Ю. Налютин, С. В. Синицын. — М. «Академия» баспасөз орталығы, 2014ж. — 272 с.

ISBN 978-601-333-250-5 (каз.) ISBN 978-5-4468-0562-4 (рус.)

Бастапқы кәсіптік білім берудің федералдық мемлекеттік стандартына сәйкес мынадай мамандықтарға арналып шығарылған: 230111 «Компьютерлік желілер», ОП.04 «Операциялық жүйелер», 230113 «Компьютерлік жүйелер және жиынтықтар», ОП.07 «Операциялық жүйелер және орталар», 230115 «Компьютерлік жүйелерде бағдарламалау», ОЖ.01 «Операциялық жүйелер», 230401 «Ақпараттық жүйелер (сала бойынша)», ОЖ.02 «Операциялық жүйелер», 230701 «Қолданбалы информатика (сала бойынша)», ОП.07 «Операциялық жүйелер және орталар».

ОЖ басқаруындағы негізгі объектілері — файлдар, пайдаланушылар және тапсырмалар туралы негізгі ақпараттар жазылған. Пайдаланушының алға қойылған тапсырмаларын логикалық орындау кезегін анықтайтын операциялық жүйе алдына қойылатын талаптар қарастырылған. ОЖ UNIX және WINDOWS көптеген пайдаланушылар жұмысын қамтамасыз етуге ерекше назар аударылады — пайдаланушыларды сәйкестендіру мәселелері, олардың жеке деректерін орналастыру, файлдар мен каталогтарға пайдаланушының қолжетімділігін басқару қарастырылған, қолжетімділік құқығымен жұмыс істеу үшін BASH тілдік құралы анықталған. Пайдаланушылардың тіркеу жазбаларын басқару тәсілдері, сонымен қатар, UNIX жүйелерінде сеанстар инициализациясы файлдар көмегімен пайдаланушылар сеанстарын дербестендіру нұсқаулығы сипатталған. UNIX-ұқсас операциялық жүйелерде және WINDOWS операциялық жүйелерінде С тіліндегі қолданбалы бағдарламаны құрастыру тәсілінің қысқаша сипаттамасы берілген.

Орта кәсіби білім беру мекемелерінің студенттеріне арналған.

ӘОЖ 681.3.066(075.32)
КБЖ 32.973-018.2я723

ISBN 978-601-333-250-5 (каз.)
ISBN 978-5-4468-0562-4 (рус.)

Құрметті оқырман!

Бұл оқулық 230000 «Ақпараттандыру және есептеу техникасы» кеңейтілген тобының барлық мамандықтарына арналған оқу-әдістемелік жиынтықтың бір бөлігі.

Оқулық жалпы кәсіби пәндерді ОЖ.04 мамандықтары үшін 230111 «Компьютерлік желілер», ОП.07 мамандықтары үшін 230113 «Компьютерлік жүйелер және жиынтықтар», ОП.01 мамандықтары үшін 230115 «Компьютерлік жүйелерде бағдарламалау», ОП.02 мамандықтары үшін 230401 «Ақпараттық жүйелер (сала бойынша)», ОП.07 мамандықтары үшін 230701 «Қолданбалы информатика (сала бойынша)» оқуға арналған.

Жаңа топтың оқу - әдістемелік жиынтығы жалпы білім және жалпы кәсіби пәндер мен мамандық бойынша модульдерден тұратын дәстүрлі және инновациялық оқу материалдарынан тұрады. Әр жиынтық жалпы және кәсіби құзыретті меңгеруге қажет, сонымен қатар жұмыс беруші талаптарын ескере отырып оқулықтар, оқу нұсқаулықтары, бақылау және оқыту жиынтықтарынан тұрады.

Оқу баспалары электрондық білім беру қорларымен толықтырылу үстінде. Электрондық қорлар интерактивті жаттығулар мен жаттығу құрылғылары бар теориялық және тәжірибелік модульдерден, мультимедиялық нысандар, Интернеттегі қосымша материалдар мен ақпараттарға сілтемелерден тұрады. Олардың ішінде терминологиялық сөздік пен оқу процесінің негізгі параметрлері тіркелетін электронды журнал болады. Зерттеу барысына жұмыс уақыты, бақылау және тәжірибелік жұмыстардың нәтижелері тіркеледі. Электрондық ресурстар оқу процесіне оңай кіріктіріледі және әртүрлі оқу бағдарламаларына тез бейімделеді.

Кіріспе

Операциялық жүйе (ОЖ) — пайдаланушыларға қолданбалы бағдарламаларды орындауға арналған орта және оларды басқару мүмкіндігін беретін бағдарламалық жиынтық, сонымен қатар, қолданбалы бағдарламаларға қолжетімділік құралдарын және аппараттық қорлар мен өңделіп отырған деректерді басқару мүмкіндігін береді.

ОЖ пайдаланушылары өз жұмыстарында ОЖ ядросы тікелей берген құралдарды немесе ОЖ басқаратын қолданбалы бағдарламаларды қолданады. Пайдаланушы өз тапсырмасын шешу үшін ОЖ ұсынатын кірістік тілде тапсырма сипаттамасын жазады, немесе басқа пайдаланушылар – бағдарламашылар жазған бағдарламаларды қолданады.

Бұндай тілдердің синтаксисі және семантикасы олардың көмегімен шешілетін тапсырмаларға байланысты әртүрлі болады. Тапсырмалар шартты түрде мынадай топтарға бөлінеді:

- ОЖ функционалдығының кеңеюі;
- ОЖ жұмыстар режимінің конфигурациялауы;
- Қолданбалы бағдарламаларды өңдеу;
- Дайын бағдарламалар көмегімен қолданбалы тапсырмаларды шешу.

Операциялық жүйемен қандай да бір байланысу тілін пайдаланатын ОЖ пайдаланушылар төмендегідей жіктеледі (сурет - В.1):

- жүйелік бағдарламашылар;
- жүйелік әкімшілер;
- қолданбалы программисттер (бағдарламашылар);
- қолданбалы

пайдаланушылар.

Сурет - В.1. ОЖ пайдаланушылар тобы

Бір тіл әртүрлі мақсаттарды шешу үшін пайдаланылады. Егер пайдаланушыларды қолданыстағы тілдеріне және орындайтын тапсырмаларына қарап толықтай жіктейтін болсақ, «байланыс тілі» — «шешілетін тапсырмалар» әр жұбы пайдаланушының рөлін оның операциялық жүйемен жұмысы кезінде анықтайды.

Кесте - В.1 операциялық жүйе пайдаланушы рөлінің негізгі типін құрайды.

Кесте - В.1. Пайдаланушылар рольдері

Нөмір	Роль	Тапсырмасы	Кіріс тілі
1	Желілік бағдарламашы	операциялық жүйе қызметтерін кеңейту	Төмен деңгейлі тілдер, оның ішінде ассемблер
2	Желілік әкімші	операциялық жүйені конфигурациялау және пайдаланушыларды тіркеу	Конфигурациялық файлдар форматтары және әкімшілендіру құралдарын басқару тілдері
3	Оператор	Жүйені ағымдағы әкімшілендіру, бағдарламалық қамтылымды орнату/өшіру, оны баптау	Конфигурациялық орнататын файлдар және орнатылатын бағдарламалық қамтылымның форматтары
4	Аппараттық қамтамасыз ету маманы	Аппаратураға қызмет көрсету, оны пайдалануға енгізу, пайдаланудан шығару	Құрылғыны күйге келтіру құралдарының және нақты ОЖ пайдалану тілдері
5	Қолданбалы бағдарламашылар	Қолданбалы тұтынушының тапсырмаларын орындауға арналған бағдарламалық қамтымды ойлап шығару	Жоғарғы деңгей тілдері, ОЖ ядросын жүйелік шақыру интерфейсі
6	Деректер Стратор әкімшісі	Жүйе деректерін мұрағаттау, ақпараттық қорларды басқару (деректер қоры, анықтамалар)	Қолданыстағы бағдарламалық құралдардың басқару және конфигурация тілдері
7	Қолданбалы тұтынушы	Дайын бағдарламалық қамтылым көмегімен нақты қолданбалы тапсырмаларды шешу	ОЖ тапсырмаларын басқару тілдері, бағдарламалық құралдар пайдаланатын басқару тілдері

Бұл оқулық ең алдымен орта білім беретін мекемелердің студенттеріне, қолданбалы бағдарламашыларына және операциялық жүйелердің пайдаланушыларына арналған, алайда ол ОЖ ортасында бағдарламалау және әкімгерлік мәселелерін де қозғайды.

Оқулықта айтылған ақпараттардың көп бөлігі UNIX және Windows буының операциялық жүйелерінде бар жалпы механизмдер мысалында беріледі. Материалдардың жарты бөлігі DOS тобының операциялық жүйелері үшін де қолданылады (Windows арналған тапсырмалар бөлімі).

Негізгі оқу операциялық жүйе ретінде кең таралған ОЖ Linux және Windows 7 операциялық жүйесі қарастырылады, дегенмен, кітаптағы материалдардың көп бөлігі UNIX-жүйесі және NT тобының Windows барлық негізгі нұсқаларына қолданылады.

Оқулық үш бөлімнен және қосымшадан тұрады:

- кіріспе терминологиясы алдағы тарауларда айтылатын ақпараттың барлығына сүйенетін операциялық жүйелердің жұмыс жасау қағидасы мен негізгі түсініктерінің анықтамасынан тұрады;
- 1 — 6 тараулар оқырманға операциялық жүйелер сүйемелдейтін объектілердің: жады, файлдар, пайдаланушылар, тапсырмалар және олармен жұмыс жасау тәсілдері даму тарихының деңгейлері туралы түсінік қалыптастырады;
- 7 — 10 тараулар оқырманның файлдар мен пайдаланушылар туралы білімін тереңдетеді, ОЖ әкімшіліктендіру мәселелерін қозғайды. Сонымен қатар UNIX және Windows операциялық жүйелер сүйемелдейтін процесстердің бір-бірімен байланысу және оларды басқару механизмдерін қарастыру сұрақтары енгізілген;
- Қосымша «Білімді бақылау» жүйесі каталогының құрылымы бойынша және UNIX операциялық жүйелердің негізгі командалары бойынша анықтамалық ақпараттан тұрады.

Оқулықтың әр тарауы өзін-өзі тексеруге арналған бақылау сұрақтарымен аяқталады.

Кітап көлемінің шектеулігінен кітапта мынадай сұрақтар қарастырылмаған:

- ОЖ генерациясы (ядро конфигурациясы, әкімшілендіру) [3];
- графикалық интерфейстер (X Window System) [2];
- бағдарламалардың жиі қолданылатын қолданбалы пакеттері;
- гетеорогенді жүйе және ортаны ұйымдастыру сипаттамасы [17];
- ядро, құрылғы драйверлерінің қосымша құрауыштарын жасау [16];

- нақты уақыт жүйесінде операциялық жүйелердегі бағдарламалаудың сипаттамалары [9];
- микроядерлік архитектуралар [15];
- әртүрлі байланыс хаттамалары негізінде желілік қосымшалар жасау [10].

Бұл тақырыптардың барлығын қызығушылық танытқан оқырманға өз бетімен зерттеу ұсынылады.

1 тарау

ТЕРМИНОЛОГИЯЛЫҚ КІРІСПЕ

1.1. НЕГІЗГІ ТҮСІНІК

Пайдаланушы операциялық жүйе ортасында өз тапсырмаларын шешкен кезде деректерді және оларды өңдеуге арналған аспаптық (бағдарламалық) құралдарды анықтау керек. Пайдаланушының тапсырмасын орындау көп жағдайда бірнеше құралдарды кезегімен пайдалануға алып келеді (мысалы, деректерді енгізу, саралау, біріктіру, шығару).

Операциялық жүйе пайдаланушыға құралдардың бастапқы жинағын және деректерді сақтау ортасын береді, сонымен қатар құралдарды пайдалану кезегін басқару құралымен қамтамасыз етеді. Пайдаланушы ОЖ ұсынатын құралдарды пайдаланып, бір немесе бірнеше кезектегі тапсырмаларды орындайтын уақыт интервалы *сеанс* деп аталады.

Кез-келген сеанс алдында пайдаланушы өзін сәйкестендіреді, соңында сеанс аяқталғанын көрсетеді. Бағдарламалық құрал-саймандарды кезекпен пайдаланудың сипаттамасы *тапсырма* деп аталады, ал тілдің өзі — *тапсырмаларды басқару тілі*.

Көптеген операциялық жүйелерде тапсырмаларды орындау командалық интерпретатормен жүргізіледі, оның толық анықтамасы алдағы тақырыптарда беріледі.

Әдетте пайдаланушыға командалық интерпретатормен байланыс жасау үшін қандай да бір интерфейс беріледі, оны пайдаланғанда командалар пернетақтадан енгізіледі, ал олардың нәтижесі экранға шығарылады. Осындай интерфейс терминалдың логикалық түсінігімен байланыстырылады — кіріс құрылғысы (қарапайым пернетақтаның) мен шығыс құрылғысының (мәтіндік ақпаратты шығаратын дисплей) жиынтығы. Қазіргі уақытта пайдаланушының графикалық интерфейсі көп қолданысқа ие (GUI), ол бұл оқулықта қарастырылмайды [2].

Бағдарлама (жалпы жағдайда) — дискте сақталатын (немесе басқа

ақпарат жинақтаушыда) процессор нұсқаулығының жинағы. Операциялық жүйе бағдарламаны орындауға жіберу үшін бағдарламаның орындалу ортасын қалыптастыруы керек — шешілетін мәселенің *ақпараттық айналасы*. Осыдай кейін операциялық жүйе орындалатын кодты және бағдарлама туралы ақпаратты оперативтік жадыға орналастырады да бағдарламаның орындалуына бастамашылық етеді.

Операциялық жүйе аппараттық ресурстарды басқару қызметін атқарады, пайдаланушы орындап жатқан бағдарламалар арасында оларды орналастырады және ішінде бағдарламаларға қажетті барлық ақпараттар қамтылған орындалу ортасын қалыптастырады. Бұндай орта бұдан кейін ақпараттық орта деп аталатын болады. Ақпараттық ортаға операциялық жүйе өңдейтін, бағдарламаның орындалуына ықпал ететін, яғни пайдаланушының тапсырмаларын орындайтын деректер мен объектілері кіреді. Бұдан әрі оқыту барысында әр түрлі сипаттағы ақпараттық орталарға мысал келтіріледі.

Бағдарлама, деректер және ақпараттық орта түсініктерін пайдалана отырып, ОЖ ортасында ақпараттық ортаның ажырамас бөлігі болып саналатын бағдарламалар мен деректер жиынтығы ретінде *тапсырмалар* түсінігін анықтауға болады.

Орындалу үстіндегі бағдарлама процесті қалыптастырады. *Процес (процесс)* ақпараттық орта мен бағдарлама деректері, орындалатын код сақталатын жадтың бөлігі арасындағы жиынтық. Әдетте операциялық жүйемен басқарылатын жадта бір мезетте процестердің бірнешеуі қатар орындала алады.

Бір процессорлы компьютерлерде тек бір процестің бағдарламалық коды қатар орындалатыны қалыпты жағдай, сол себептен процестердің бір бөлігі күту режимінде ал басқа бір процес орындалу үстінде болады. Процестер осылайша кезек құрады, операциялық жүйе кезектегі бірінші процеске басқаруды береді, одан кейін келесіне және с.с.

Пайдаланушыдан пернетақта арқылы кіріс деректерін ала алатын және өзінің жұмыс нәтижесін экранға шығара алатын әлеуеттік мүмкіндігі бар процес алдыңғы жоспар процесі; тікелей пайдаланушы ықпал етпей орындалатын процес — фондық процес деп аталады.

Өздерінің жұмыс барысында процестер процессордың есептеуіш қуатын, оперативті жадын пайдаланады, операциялық жүйенің сыртқы файлдарына, ядроның ішкі деректеріне жүгінеді. Бұл объектілердің барлығы процестің ақпараттық айналасына кіріп, ресурс деп аталады.

Ресурс ОЖ қол жетімділігін жасайтын физикалық объект болуы мүмкін — процессор, оперативті жады, дискті жинақтауыштар, сондай-

ақ тек ОЖ айналасында ғана жүзеге асатын логикалық объект бола алады, мысалы орындалатын процестер кестесі немесе желілік қосылу процесі. ОЖ тарапынан ресурстарды басқару қажеттілігі бірінші кезекте ресурстардың шектеулігі нәтижесінде туындайды (көлемі, пайдалану уақыты бойынша, қызмет көрсетіліп отырған пайдаланушылар санына байланысты және т.б.). Бұндай жағдайда операциялық жүйе ресурстардың сарқылуының алдын алып олардың шектелулерін басқарады немесе ресурстың сарқылуына байланысты жағдайларды өңдеу құралын ұсынады. ОЖ берілген көптеген ресурстар шектеулерінен кейін жүйе әкімшісімен өзгертіле алады.

Егер операциялық жүйе бірнеше процеске бір мезетте ресурсты пайдалануға рұқсат берген жағдайда оның ресурстарын 1.1-суретте [12] көрсетілген түрлерге бөлуге болады.

Бөлінбейтін ресурстарды берілген уақыт бөлігінде тек бір процес пайдалана алады, бұл кезде басқа процестердің ресурсты пайдаланып отырған процес оны толық босатпайынша оған қолжетімділіктері жоқ. Бұндай ресурсқа шектелген режимде жазба жасау үшін ашылған файлды мысал ете аламыз. Басқа процестердің бұл файлды пайдалануға жасалған әрекеттері (тіпті оқу үшін) сәтсіздікпен аяқталады.

Бөлінетін ресурстар бірнеше процеспен бір уақытта қолданыла алады. Сонымен қатар бұндай ресурстарға басқа процестер қолжетімді (мысалы, ағымдағы жүйелі уақытты анықтайтын сағаттар) болады.

Кейбір бөлінетін ресурстар бір мезеттік қол жетімділікті қамтамасыз ете алмаса да ресурс толық босаған уақытын күтпестен оларды бірнеше процестердің қолданылуына мүмкіндік береді.

Сурет - 1.1. операциялық жүйеде қолжетімді ресурс түрлері

Бұл жағдайда ресурсты уақыт бойынша бөлу үшін сәттерді кванттау қолданылады.

Уақыттың әр квантында бір процес осы ресурсты иемденуге толық

және шектеулі құқыққа ие болады. Оған қоса осы ресурс пайдаланып отырған процес кезінде квант мәні толық уақыттан, яғни процеске пайдаланушы тапсырмасын орындауға қажетті уақыт аралығынан әлдеқайда аз.

Уақыт бөлінісі қолжетімді ресурс мысалына көп мақсатты ОЖ процессорлық уақыты қызмет ете алады. Әр уақыт квантында бір процес нұсқаулығының белгілі бір саны орындалады, одан кейін басқа процеске бұйрық келеді де оның нұсқауларының орындалуы басталады.

Бөлінетін ресурсқа қол жетімділік күтіп тұрған процестер басымдылықпен кезекке тұрады. Дегенмен кейбір процестердің маңыздылығы жоғары болуы себебінен ресурсқа қол жетімділік рұқсатын тез алады.

1.1.1. Операциялық жүйенің типтік құрылымы

Әдетте операциялық жүйе құрамында екі деңгейді ерекшелейді: жүйе ядросы және көмекші жүйелік бағдарламалық құралдар, олар кейде жүйелік утилита деп аталады. *Ядро* жүйе ресурстарын басқару бойынша барлық қызметтерді ақтарады — физикалық және логикалық, сонымен қатар пайдаланушыларға (пайдаланушылар бағдарламасы) осы ресурстарға қол жетімділікті бөледі. Жүйелік бағдарламалық қамтылым көмегімен пайдаланушы ядро ұсынатын құралдарды басқарады.

Қарапайым операциялық жүйе ядросына келесі құраушылар кіреді: пайдаланушы сеансын басқару жүйесі, файлдық жүйе, тапсырмаларды (процестерді) басқару жүйесі, енгізу және алып тастау жүйесі. Қолданбалы бағдарламалармен ОЖ ядросының интерфейсі жүйелік шақыру бағдарламалық интерфейс көмегімен, ал аппараттық қамтылым бар интерфейс — драйвер көмегімен (Сурет -1.2) орындалады.

Пайдаланушылар сеанстарын басқару жүйесі пайдаланушы сеансын оның ОЖ жұмыс бастаған кезде тіркейді, сеанстың ақпараттық ортасына кіретін оперативті ақпаратты сақтайды, кіріс-шығыс жүйесі көмегімен пайдаланушы терминалын шынайы немесе виртуал құрылғымен сәйкестігін қолдайды, пайдаланушының жүйемен жұмысы аяқталған кезде сеансты сыпайы аяқтайды.

Сурет - 1.2. Қарапайым операциялық жүйе ядросының құрылымы

Файлдық жүйе сыртқы сақтау құрылғыларында сақталатын деректерді түрлендіруді орындайды (мысалы, дискті жинақтауыш немесе flash-жинақтауыш), логикалық объекттер файлдар және каталогтар. Сонымен қатар файлдық жүйе сеанстарды басқару жүйесі тарапынан ұсыныс түссе немесе файлдық жүйені жүйелік шақыртулар интерфейсі арқылы пайдаланған кезде файлдар мен каталогтарға қол жетімділікті шектейді.

Процестердің басқару жүйесі ресурстарды орындалатын тапсырмалар (процестер) арасында таратады, процестер жинақтауышын басқа процестер модификациясынан қорғанысты қамтамасыз етеді, процес аралық өзара байланыс механизмдерін жүзеге асырады.

Кіріс-шығыс жүйесі ядроның жоғарыда көрсетілген барлық құраушыларының өтінімдерін өңдейді және оларды ОЖ қолдайтын логикалық құрылғылар шақыртуына түрлендіреді. Бұндай құрылғылардың әр қайсысы логикалық объект болып табылады, оған жүгіну ОЖ үшін стандартты құралы арқылы (мысалы, оперативті жадының адресіне немесе арнайы файлға) жүреді. Логикалық құрылғы

толықтай виртуалды болуы мүмкін (ОЖ ядросының ішінде толықтай жұмыс жасайды немесе нақты аппараттық құрылғылармен драйвер арқылы байланысқан логикалық объектті білдіреді.

Толықтай виртуалды құрылғы ретінде UNIX-жүйелеріндегі «қара құрдым» /dev/null келтіруге болады. Бұл құрылғыға жазылатын барлық ақпарат жоғалып кетеді, яғни олар тапсырмаларды шешуге арналған болмашы деректерді жоюға қолданылуы мүмкін.

Құрылғылар драйверлері — бұл аппараттық құрылғылар үшін басқаратын командалар кезегімен жүйенің кіріс/шығыс сұрауларын түрлендіретін жүйелі бағдарламалар. Әр құрылғының драйвері оның аппараттық жүзеге асуының ерекшеліктерін жасырады және кіріс/шығыс жүйесіне жүйенің аппараттық қамтамасыз етуіне қолжетімділік стандартталған интерфейсін ұсынады.

Қолданбалы бағдарламашы тұрғысынан қарағанда ОЖ ядросы құраушыларына қолжетімділік жүйелік шақыртулар интерфейсі көмегімен жүзеге асырады — стандартталған міндеттер жинағынан тұратын кітапханалар жинағы. Бұндай жинақтың әрқайсысы қолданбалы тапсырмаларды шешуге арналған: желілік ресурстарға, графикалық режимге қолжетімділік, процес аралық өзара байланысты жүзеге асыру және т.б.

1.1.2 Операциялық жүйелердің жіктемелері

Ядроның құраушы бөліктерінің күрделілігі және олар жүзеге асыратын қызмет түрлері бірінші кезекте бір сәтте ОЖ қызмет көрсетілетін пайдаланушылар санына және бір мезгілде қатар орындалатын процестер санына тәуелді. Сол себепті ОЖ жіктелуін осы екі параметр бойынша жүргізіп ядро құраушыларын ОЖ әр типінде қарастырамыз.

Бір мезетте қызмет көрсетілетін пайдаланушылар санына байланыста операциялық жүйелер бір пайдаланушылы (бір мезетте бір пайдаланушының сеансынан артық қабылдамайды) және көп пайдаланушылы (бір мезетте бірнеше пайдаланушыны сүйемелдейді). Көп пайдаланушылы жүйелер пайдаланушы деректерін басқа пайдаланушылардың санкцияланбаған қолжетімділіктерінен қорғанысын қамтамасыз етіп қана қоймай бірнеше пайдаланушыларға ортақ мәліметтерді бөлу құралын ұсынады.

ОЖ бұл түрлерінің ерекшеліктерін толықтай қарастырайық.

Бір мезетте орындалатын процестер санына байланысты операциялық жүйелер бірмақсатты (жұмыс істейтін процес саны біреу) және көпмақсатты (жұмыс істейтін процестер саны көп) болып бөлінеді. Көпмақсатты жүйелердің бірмақсатты жүйелерден негізгі айырмашылығы ресурстарға қолжетімділікті басқару құралының бар болуы — ресурстарды бөлу және қолданыстағы ресурстарды блоктау.

Бір пайданалануышылы ОЖ. ОЖ бұл түрі пайдаланушының тек бір сеансын сүйемелдеуді қамтамасыз етеді. Пайдаланушы жұмысының жаңа сеансы тек алдыңғы сеанс біткен соң ғана басталай алады. Бұл жағдайда пайдаланушының жаңа сеансында дәл сол ақпараттық орта сақталады.

Бірпайдаланушы ОЖ тарапынан қарағанда пайдаланушыларды ажырату мүмкін емес, сол себепті егер бұндай операциялық жүйемен бірнеше пайдаланушы жұмысын бастаса, ол әрқайсысына барлық ресурстарға қолжетімділікті қамтамасыз етеді және сол ақпараттық ортаның өзіне де қолжетімділікті қамтамасыз етуі ықтимал. Пайдаланушы өзінің бірегей деректерімен, мысалы салынбалы дисктегі мәліметтерімен жұмыс жасай алады. Осындай жұмыс жағдайында жұмыс сеансының ақпараттық ортасы жүйеде әртүрлі болады.

Бір пайдаланушылы ОЖ сеанстарымен басқару жүйесі құрамына тек бастамалау құралы және пайдаланушының ақпараттық ортасын қолдау құралы кіреді. Бұған қоса көптеген бір пайдаланушылы ОЖ (мысалы, DOS) пайдаланушы сеансын бастапқы жүктемелеу сәті бірден ядроны және сценарийлерді жүктегеннен кейін басталады.

Сеансты аяқтау сәті ОЖ ядросын жадыдан шығару сәтімен сәйкес келеді (ОЖ жұмысын аяқтар алдында немесе құрылғыны тоқсыздандыру нәтижесінде). Осылайша, пайдаланушы сеансының өмір сүру уақыты ОЖ шамамен жүйенің жұмыс жасап тұрған уақытына тең.

Пайдаланушыларды ажырата алмау салдарынан сеанстарды басқару жүйесі және файлдық жүйе айтарлықтай шамада жеңілдетіледі. Бірпайдаланушылы ОЖ сеанстарын басқару жүйесінің ішіне пайдаланушылардың идентификациясы мен аутентификациясы құралдары кірмейді, сонымен қатар олардың сеанстарының ақпараттық ортасын қорғау құралдары да жоқ. Бірпайдаланушылы ОЖ файлдық жүйесіне ереже бойынша файлдар мен каталогтарға қолжетімділікті шектеу күрделі механизмдері жатпайды, бірақ файл жүйелерінде файлдар мен каталогтармен және олардың атрибуттарымен жұмыс режимін білдіретін жалғаулар болуы мүмкін.

Операциялық жүйенің тек бір пайдаланушының жұмыс сеансын қолдауы бір мезгілде пайдаланушының бірнеше тапсырмасын орындау

мүмкіндігін жоққа шығармайды. Басқа сөзбен айтқанда бірпайдаланушыға операциялық ондық жүйе көпмақсатты болуы мүмкін.

Көппайдаланушылы ОЖ. ОЖ бұл түрі көп пайдаланушылардың бір мезгілдегі жұмысын қатар қамтамасыз етеді, файлдық жүйемен және сеанстарды қолдау жүйесімен жүзеге асатын атқарымдар жинағын біршама кеңейтеді. Көптеген пайдаланушыларды қолдау төмен деңгейде кіріс/шығыс жүйесінде және процестерді басқару жүйесінде көрінеді.

Пайдаланушылардың сеанстарын басқару жүйесінің құрамына пайдаланушыларды идентификациясы және аудентификациясының құралы кіреді. Ол әр сеансты нақты және виртуал терминалмен байланысын қамтамасыз етеді, сеанстың бастапқы ақпараттық ортасының инициализациясы құралынан тұрады және сеанс деректерін қорғауды қамтамасыз етеді.

Көппайдаланушылы ОЖ файлдық жүйесі файлдар мен каталогтарға қолжетімділіктерді шектеуді қамтамасыз етеді, оған негіз сеанстарды басқару жүйесінен алынған пайдаланушылар идентификаторы. Файлдық жүйеде әр файл мен каталог оған пайдаланушылар қолжетімділік құқығын анықтайтын ақпараттық бұғатпен бірге жүреді. Пайдаланушыға дерек тек өзіне ғана қолжетімді болатындай, файлдар мен каталогтардағы деректерді басқа пайдаланушылар тек өзгерту емес, тіпті оларды оқи алмайтындай етіп пайдаланушы құқығын анықтау мүмкіндігін ұсынады. Дегенмен бір ақпаратқа ортақ қолжетімділік қажеттілігі туындағанда бір ақпарат жинағына бірнеше пайдаланушылардың оқуы немесе жазу мүмкіндігі ашылу ықтимал.

Кіріс/шығыс буферленуі мен құрылғыларға тікелей қосылудан басқа көп пайдаланушысы бар ОЖ кіріс/шығыс жүйесі пайдаланушылардың құрылғыларға қолжетімділігін бөлуді басқарады, яғни құрылғылармен бөлінетін реурстар секілді басқарады.

Әдетте көп пайдаланушылы ОЖ бірнеше пайдаланушылардың көп бағдарламасын бір мезгілде жүзеге асыруын қамтамасыз ететіндіктен олар көпмақсатты болып та саналатынын ескерген жөн.

Бірмақсатты ОЖ. Операциялық жүйелердің бұл тобы бір уақытта тек бір тапсырманы орындауға арналған. Жүйе басталғаннан кейін басқару бірден пайдаланушы жұмысы үшін жабын рөлін атқаратын бағдарламаға беріледі. Ереже бойынша бұндай жабынның қызметтерінің бірі — басқа бағдарламаларды іске қосу.

Бағдарлама іске қосылмас бұрын ақпараттық ортаның жабыны сақталады. Бағдарлама іске қосылған соң оның процесіне толықтай басқару және барлық ресурстарға қолжетімділік беріледі. Бағдарлама

аяқталған кезде процес жады босайды, ақпараттық орта жабыны қайта қалпына келеді және операциялық жүйе арқылы оған басқару қайтарылады.

Бағдарламалардың іске қосылуы бұндай ОЖ үшін кезекті. Егер бір бағдарламаға орындалуды жүзеге асыру үшін басқа бағдарламаны шақыру қажет болса, шақырып отырған бағдарламаның ортасы сақталады және шақырылған бағдарлама аяқталған соң бағдарламаның ақпараттық ортасы қайта қалпына келеді.

Біртапсырмалы ОЖ кіріс/шығыс жүйесінің құрамына құрылғыларға қолжетімділікті бөлу құралдары кірмейді, өйткені құрылғыны бір мезгілде тек бір процес қана пайдаланады.

Біртапсырмалы ОЖ көптапсырмалы да бола алады. Бұндай жүйенің мысалына пакеттік өңделуі бар ОЖ бола алады. Осындай ОЖ пайдаланушылар бағдарламаларды орындауға тапсырмалар кезегін құрады, бұған қоса тапсырмалар бірнеше пайдаланушыларға тиесілі болулары мүмкін. Жүйе пайдаланушылар бағдарламасын кезекпен орындайды, оның үстіне пайдаланушыны ауыстырар алдында бұған дейінгі пайдаланушының жұмыс сеансы аяқталады да жаңа пайдаланушы сеансы басталады. Осылайша, тапсырма өзгерген сайын әр бағдарламаның ақпараттық ортасы ауысады.

Көпмақсатты ОЖ. Көпмақсатты ОЖ бір уақыт мезгілінде көптеген бағдарламалар (процестер) іске қосыла алады. Бұл жағдайда процестерді басқару жүйесінің құрамына процестерді жоспарлау кіреді, ол өз кезегінде мынадай қызметтер атқарады:

- Процестерді жасау және жою — бағдарламаны жадқа жүктеу, ақпараттық орта жасау және ол жаңадан пайда болған кезде процеске басқаруды беру, ақпараттық ортаны өшіру және ол жойылған кезде жадыдан процесті шығару;
- Процестер арасында жүйелік ресурстарды бөлу — процестердің орындалуын жоспарлау, процестер кезегін қалыптастыру және кезектегі процестер артықшылықтарын басқару;
- Процес аралық байланыс — жалпыға ортақ деректерді процестер арасында бөлу немесе басқарушы өзара байланыстарды бір мезгілде орындалатын процестер арасында жіберу;
- Процестердің орындалу синхронизациясы — кейбір шарттар орындалмайынша процестердің орындалуын уақытша тоқтату, мысалы бір процеспен басқарылатын әсер етуді қайта жіберу.

Мұндай ОЖ кіріс/шығыс жүйесі қиындайды, өйткені кез-келген ресурс (файл немесе құрылғы) бір процеспен бірге пайдаланылуы мүмкін. Қолжетімділік шиеленісінің алдын алу үшін бұғаттау механизмі пайдаланылады, ол бөлінбейтін ресурсқа қолжетімділікті бір

мезгілде тек бір процеске рұқсат етеді.

UNIX тобының ОЖ көп пайдаланушылы көптапсырмалы операциялық жүйеге жатады. Осы себептен операциялық жүйенің осы тобы оқулықтың негізі болып алынды.

Жоғарыда айтылғандай оқулық қолданбалы бағдарламашыларға және пайдаланушыларға арналғандықтан, ол жерде операциялық жүйе әзірлеу және қолданбалы бағдарламалық қамтамасыздандыру ортасы ретінде қарастырылады.

Оқулықта тек UNIX ОЖ бастапқы құралы ғана қарастырылады, және мысалы X Window System B графикалық амалдары секілді қандайда бір кеңейтілулерге назар аударылмайды.

1.2. МАМАНДАНДЫРЫЛҒАН ЖӘНЕ ӘМБЕБАП ОПЕРАЦИЯЛЫҚ ЖҮЙЕЛЕР. НАҚТЫ УАҚЫТТЫҢ ОПЕРАЦИЯЛЫҚ ЖҮЙЕЛЕРІ

Жүйедегі пайдаланушылар санының негізінде және бір мезгілде орындалатын процестер санына байланысты жіктелуінен бөлек операциялық жүйелердің тағы бір жіктелу түрін қосуымызға болады: жалпы тағайындалудың операциялық жүйесі және арнайы тағайындалудың операциялық жүйесі.

Жалпы тағайындалудың операциялық жүйесі классына бірпроцесті және көппроцесті, бір пайдаланушылы және көп пайдаланушылы бола алатын жүйелер. Бұл операциялық жүйелер қарапайым үстел үстілік жүйелер қатарында жұмыс жасайды. Олардың негізгі тағайындалуы жүйенің пайдаланушыларына есептеу жүйелерінің аппараттық құралдарын басқарудың қолайлы және түсінікті механизмін ұсыну болып табылады. Пайдаланушыны төмен деңгейлі операциялар мен интерфейстерден барынша оқшаулап, оның өтінімін өңдеумен айналысады. Осындай операциялық жүйелер бірінші кезекте қолдану қарапайымдығына бағытталған, өйткені олардың басты пайдаланушылары бағдарламалаушылар емес, кәсіби біліктілігі төмен немесе орташа пайдаланушылар.

Көбінесе мұндай пайдаланушылар анимацияланған түсқағаздардың әдемі суретінің және графикалық интерфейстің барлық мүмкін әдемілігінің артында компьютердің барлық аппараттық жиынтығын басқаратын қуаты мықты бағдарламалық жиынтық жасырулы екенін түсіне бермейді. Ұқсас операциялық жүйелерге Windows, Linux, Apple iOS және т.б. топтарының үстелдік нұсқасы операциялық жүйелері

жатады. Басқаша айтқанда бұл операциялық жүйені пайдаланушы жұмыста қандай да бір мақсатты шешу үшін пайдаланса үйде де ойын-сауық құралы ретінде пайдалана алады.

Жалпы тағайындалудағы операциялық жүйесінен кереғар *арнайы тағайындалған ОЖ* күнделікті өмірде өте сирек қолданылады. Бұл операциялық жүйелердің негізгі пайдаланушылары— кәсіби біліктілігі жоғары бағдарламашылар. Ұқсас операциялық жүйелер арнайы есептеу жүйелерінің ресурстарын басқаруға арналған. Көп жағдайда бұл жүйелер кіріктірілген болып саналады, яғни өздері басқаратын құрылғыға тікелей кіріктірілсе де жұмыс істеуі қажет жүйелер болып есептеледі. Оларға Android, iOS, Windows CE және т.б. операциялық жүйелері жатады.

Арнайы тағайындалған операциялық жүйелердің қосалқы жиынтықтарының бірі *нақты уақыт операциялық жүйелері* болып саналады. Көртеген кіріктірілетін жүйелер мұндай бағдарламалық-аппараттық жиынтық құрамында жұмыс істейтін операциялық жүйе, сыртқы оқиға және ішкі деректерге өте аз уақыт ішінде әрекет етуін талап етеді. Басқа сөзбен айтқанда нақты уақыттың операциялық жүйелері — бұл белгілі бір уақыт ішінде талап етілетін қызмет деңгейін қамтамасыз ететін жүйелер. Нақты уақыт операциялық жүйелерін екі классқа бөлуге болады: қатты нақты уақыт жүйесі және жұмсақ нақты уақыт жүйесі. Талап етілген орындалу уақытынан аспай жұмыс істейтін операциялық жүйелерді қатты нақты уақыт операциялық жүйелеріне жатқызады. Егер операциялық жүйе талап етілген уақыттың жартысында ғана жұмыс жасаса жоғары уақыт шектеуін қатты сақтамаса бұл жүйені жұмсақ нақты уақыттың операциялық жүйесіне жатқызамыз. Дегенмен осы кезде нақты уақыт ОЖ екі классы үшін екіталай талабы бұл жүйелердің детерминизмі болып табылады, яғни оның оның әрекетінің болжамдығы.

Нақты уақыт жүйесі жүйе реакциясы кешеуілдеуі материалдық құралдардың жоғалуына алып келетін, адам өміріне қауіп әкелетін және т.б. апаттық жағдайларға әкелуі мүмкін жағдайларда қажет. Бұндай жүйелерде көп жағдайда операциялық жүйелерсіз өткізіледі.

Нақты уақыт операциялық жүйелерін пайдалану басқарушы БҚ жасап шығару уақытын қысқарту мүмкіндігін береді және келесі жағдайларды оның әрекетінің болжамдығын арттыруға ықпал етеді:

- Егер құрастырылған басқаратын БҚ көлемі бойынша үлкен болса;
- егер оның орындалуы барысында бірнеше есептеуші ағымдар қажет болса;
- егер шешілетін тапсырма ресурстарға қолжетімділік

синхронизациясы бойынша талаптары күрделі болса және т.б.

Бұндай операциялық жүйелерде процестерді басқару үшін екі тәсілдің бірі қолданылады:

1) Оқиғалардың артықшылықтары негізінде басқару. Бұл стратегияны пайдаланған кезде, басқару артықшылыққа ие оқиғаны өңдеуге байланысты процеске беріледі;

2) Уақытты бөлу негізіндегі басқару. Бұл жағдайда процестерді ауыстырып қосу берілген уақыт интервалында қалыпты түрде үзуге негізделіп және оқиға басталған кезде орындалады.

Жүйелердің осындай көп түрлерінде процестер құрамы өзгеріссіз болады. Олар операциялық жүйелер басталған кезде іске қосылады және оның жұмысы аяқталғанша жалғасады. Қолданылмай тұрған процестер пассивті күйге ауысуы мүмкін және егер осы процестің белсенділігі қажет оқиғаны өңдеу кезінде пассивті күйден шығады. Процестердің бұндай іске қосылу және аяқталу жүйесі операциялық жүйе әрекетінің және реакция уақыты бойынша параметрлерінің болжамдығы талабымен туындаған. Сол себепті осындай көптеген жүйелерде жаңа процестерді іске қосу жай жіберілмейді, олардың барлығы алдын ала анықталған болулары керек.

Қазіргі уақытта нақты уақыт операциялық жүйелерінің көп түрлері бар: LynxOS, RTLinux, VxWorks және т.б. нақты уақыт операциялық жүйесінің ең кең таралған түрі QNX болып табылады, ол кей кезде үстел үстілік ретінде де қолданылады. Бұл коммерциялық таралған өнімдерден басқа жеке компаниялардың өздері әзірлеген операциялық жүйелер де бар, олар коммерциялық негізде таратылмайды, тек дайын бағдарламалық-аппараттық жиынтық құрамында қолданылады.

1.3.

ОПЕРАЦИЯЛЫҚ ЖҮЙЕЛЕРДІҢ ҚЫЗМЕТІ ЖӘНЕ ОЛАРДЫҢ ДАМУ ДЕҢГЕЙЛЕРІ

Операциялық жүйелердің тарихи қызметтері есептеуші техниканың өзінің дамуымен бірге дамыды. Өз уақытында есептеуші машиналар

топтарын ерекшелеу қабылданады, олардың орындалуы үшін келесідей элементтік базаларға сүйенеді: электронды лампалар, жартылай өткізгіш транзисторлар, микросұлбалар және т.б. Дегенмен жаңа жүйелік бағдарламалық құралдардың шығуына ықпал еткен бұл қасиеттері емес.

Жалпыға арналған жүйелік бағдарламалық қамтылымдардың дамуына негізгі ықпал еткен төрт фактор бар: есептеуіш машиналар архитектурасының унификациясы, оперативті жад көлемі, процессордың тез әрекет етуі және перифериялық құрылғының құрамы. Осылайша, жұмысында негізінен құрастырушылары қатысқан алғашқы машиналар үшін ешқандай да унификация сөз болған жоқ. Әр ұжымда командалар құрамымен де есептеу разрядты ұяшықтар көлемімен де тәжірибелер өткізілді. Тапсырмамен жұмыс жасау барысында бағдарламалаушы және оператор бір адам есептеуіш құрылғының толыққанды иесі болды. Ол бағдарламаны өзі жүктеді, деректерді өзі енгізді және нәтижелерін бағалады.

Стандартты бағдарламалар кітапханасы. Сериялық шығарылатын машиналар шыққаннан кейін бағдарламалық қамтымды ойлап шығару амалын стандарттау мүмкін болды. Бағдарламалаушыларды пайдаланушылардың бөлек класына ерекшелеу орын алды. Бағдарламалау өтілін жинақтау және жалпылауға арналған құрал сол уақытта стандартты бағдарламалардың кітапханасы болды. О кітапхана келесідей типтік атқарымдарды жүзеге асырады: тригонометриялық (sin, tg, arcsin), математикалық (log, exp, sqrt), сыртқы жинақтауыштардан мәліметтерді енгізу, шығару, бір есептеу жүйесінен басқасына түрлендіру және т.б. Осындай кітапхананың әдеттегі мысалы ИЖ-2 стандартты бағдарламасының (интерпретациялайтын жүйе) М-20 машиналарына арналған кітапханасы бола алады. Сонымен бірге М-20 сериялық шығарылатын үш адрестік жүйе командалары бар машина болғанын ескерген жөн; оның командалар жүйесіне ұқсас командалар БЭСМ-3, БЭСМ-4, және кейінірек М-220 машиналарында болды.

ИЖ-2 кешенді бағдарламасының жиынтығын бағдарламалаушы оперативті жадыға негізгі функционалдық бағдарламаны жүктемей тұрып жүктейді. Орындалып жатқан бағдарламаға жүйелік кітапхананың бір атқарымына жүгінуіне мүмкіндік беретін стандартты интерфейс қолданылды. Ол үшін атқарымның қажетті коды мен өзара байланыс үшін жады аумағын көрсету керек, мысалы мәліметтерді жүктеуге арналған жады бөлігі.

Кейінірек есептеуіш атқарымдардың көп бөлігі бағдарламалау тілі кітапханасынан ауысты және тіпті мәліметтерді бір пішіннен екінші

түрге ауысты, ал жүктеу және шығарға қатысты типтік әрекеттер операциялық жүйенің ажырамас бөлігіне айналды.

Қазіргі таңда бағдарламалаушылар ортасында осындай атқарымдарға арналған операциялық орта үшін API (Application Programming Interface) термині қолданылады, ұқсас жүйелік құралдардың қолданбалы сипаттамасын ерекшелейді.

Осыған орай есептеуіш машиналардың архитектурасы типтестірілді және олардың тез әрекет етуі ұлғайған сайын, оперативті жадыны жүктеуге, мәліметтері бар жинақтауыштарды орнату, яғни тапсырманың ақпараттық ортасының инициалдануы және т.б. орындауға арналған бағдарламаларды дайындауға кететін уақытты жоғалту мәселесі қауырт болды. Шынымен де, процессор он мыңдаған операцияларды 1 секундта орындағанда, бағдарламасы бар перфокартаны негізге орнатуға және енгізуге жұмсалатын бір жарым – екі минут 300 000 – 500 000 командаларын орындау мүмкіндігін жоғалтуға әкеледі.

Пакеттік мониторлар. Шешім жүйелік бағдарламалауларды ойлап шығарумен табылды — мониторлар, бағдарламалар мен деректер пакетін алдын-ала дайындалған магниттік жинақтауыштардан (сол кезде бұл магниттік ленталар болды) тез жүктеуге мүмкіндік берді. Пакеттердің орындалуын сыртқы қызмет көрсету үшін мамандардың арнайы тобы шықты — операторлар. Олар магниттік жинақтауыштарға бағдарламалаушылар дайындаған пакеттерді жүктеді. Одан кейін мониторлық бағдарламаны іске қосты, ол өзі кезегімен оперативті жадыға кезекті тапсырманы ескерді, қажет ақпараттық ортаны жасап, бағдарламаны өзі орындауға жіберді.

Бағдарламаның тез орындалуы үшін нәтижелер бірден басылып шыққан жоқ, жинақтаушы құрылғыларға шығарылды — жиі осындай магниттік ленталарға. Тек барлық есептеу жұмыстары аяқталғаннан кейін ғана мәліметтер біртіндеп қағазға басылып шығарылды. Ол жұмыс жылдамдығы басып шығару жылдамдығынан мың есе асып түсетін процессорлардың тоқтап тұруын қысқартуға мүмкіндік берді.

Жүйе құралдарының дамуының бұл уақыты операциялық жүйе құрамына жүйелік енгізу бағдарламасын енгізумен сипатталды. Оның қатарында пакеттерді дайындауға арналған, жүйелік монитор бағдарламасы (кейде оны бағдарламаны немесе пайдаланушылар сеансын басқару бағдарламасын басқаратын диспетчер деп атады) және мәліметтер пакетін есептеу кезінде алынған қайта жүктеуге арналған деректерді шығаруға арналған құрылғы, мысалы баспаның жүйелік шығару бағдарламасы да бар.

Мониторлық пакетпен бірге бағдарламалаушылар өз

тапсырмаларын орындай алатын тіл пайда болды. Бірінші ол тапсырмаларды басқару тілі деп аталды. Кейінірек, «операциялық жүйе командаларының тілі» термині жиі қолданыла бастады, өйткені оның құраушы бөліктері ОЖ интерпретаторы командаларын орындайтын командалар болды.

Оперативті есте сақтау құрылғысының сыйымдылығы өскен сайын операторларда осы үш бағдарламалық құралдарды (енгізу, мониторингілеу және шығару) әрқашан есептеуіш құрылғы жадында ұстап тұруға мүмкіндіктері пайда болды. Нәтижесінде жұмыстың барлық кезінде жаңа тапсырмаларды енгізіп және дайын болғанда нәтижелерін басып шығару мүмкін болды. Енгізу және шығару артықшылығы төмен режимде жасалды, ал негізгі уақытта процессор тапсырмалар пакетінің тапсырмаларын шешу үшін пайдаланылады.

Тапсырмаларды басқару тілінің бар болуы бағдарламалаушыға оның шешімі кезінде қасында болмауға мүмкіндік берді. Оған есептеу кезінде орын алуы ықтимал әртүрлі оқиғаларды сипаттап, операциялық жүйенің сәйкес әрекет етуінің алдын алуы жеткілікті. Есептеудің одан кейінгі қызмет көрсетуі оператор міндетіне жүктелді. Осы уақытқа бағдарламалаудың әртүрлі тілінде есептеу бағдарламаларын құратын бағдарламашыларды осы бағдарламалар көмегімен қолданбалы тапсырмаларды шешкен, тапсырмаларды басқару тілінің көмегімен деректер пакеті мен деректерін біріктіретін пайдаланушылардан анық бөлінуін жатқызуға болады.

Мульти бағдарламалау, көп мақсаттылық. Процессор жұмысының жылдамдығының ары қарай өсуі және оперативті жады көлемінің артуы машинада бір мезгілде бірнеше пайдаланушылардың бағдарламалары орындалатындай жағдайға алып келді. Ол екі мәселені шешуді қажет етті — бір-біріне параллель орындалатын бағдарламалардың тәуелсіз орындалуын қамтамасыз ету және бағдарламалары қатар орындалатын әр түрлі пайдаланушылардың мәліметтерін қорғау.

Бірінші мәселені орындау үшін диспетчер қызметін дамыта бастады пакеттегі бағдарлама ауысымы бойынша жұмысты тек қана жоспарламай, белсенді бағдарламалардың қайсысын орталық ресурсына беруге болатыны туралы шешім қабылдайды. Оған бір мезетте қатар орындалатын бағдарламалар арасында жадыны бөлу мәселесін шешу қажет болды.

Өз кезегінде аппараттық құралдар бір процес екіншісінің жұмысына кедергі келтірмейтіндей бақылай бастады. Егер жады басқа тапсырма кеңістігіне жүгінетін болса, бағдарламаның дереу тоқтауына (үзілу) алып келетін жадыны қорғауға арналған әртүрлі

құрылғыларының есебінен қамтамасыз етілді

Көргеніміздей, пакеттік өңдеу кезінде мультибағдарламалану пайда болды. Мульти бағдарламалау — жүйелік енгізу, шығару және пайдаланушы пакет мониторы бағдарламаларының параллель орындалуы. Дегенмен жүйелік бағдарламалар бір-біріне жұмыс жасауға кедергі келтірмеуге бір-бірімен өзара «келісті».

Енді есептеуіш құрылғыда әдейі немесе кездейсоқ ақпараттық ортада басқа пайдаланушы деректерін бұзуы ықтимал әртүрлі пайдаланушылардың бағдарламалары параллель жұмыс жасады. **Деректерді басқару.** Өзге пайдаланушылар бағдарламасының оларға санкцияланбаған қолжетімділіктің алдын алатын сыртқы жинақтаушыдағы деректермен жұмыс жасаудың ережелерін ойлап табу қажет болды. Осы мақсат үшін деректерді жазуды деректер жинағы (Data set) жинақтауыштарына ұйымдастыра бастады. Бұл мәліметті кім, қашан, жасағаны туралы қосымша ақпаратпен қамтамасыз етті және оған қол жетімділікті анықтады.

Құрылғыға (ақпаратты жинақтауыш) тікелей қолжетімді пайдаланушылардың бағдарламасы деректер жинағын ұйымдастыруға қатысты келісімді бұзуы мүмкін екені анық болды. Сол себепті сыртқы құрылғылар арасында деректермен алмасу бойынша барлық операциялар операциялық жүйенің меншігіне айналды, солай жүйелік кітапхана құрамы кеңейтілді.

Егер бағдарламаға сыртқы құрылғыдан мәлімет алу қажет болса, ол сәйкес API операциялық жүйесіне жүгініп, өз жады аймағына деректердің кезекті бұғаттауын алады. Ақпарат алудың осындай ұйымдастыру тағы бір пайдалы қасиетті қалыптастырды — бағдарламалардың деректерді физикалық ұйымдастырудан тәуелсіздік. Құрылғы тек операциялық жүйемен жұмыс жасағандықтан, пайдаланушы бағдарламасы деректер жинақтауышының сипаттамасынан көп деңгейде тәуелсіз болды.

Бұдан басқа, операциялық жүйеге қолжетімділіктің *құқықтығын* тексеру қызметін өзіне алды. Пайдаланушы — деректер жинағының иесі — ол өз жинағымен жұмыс жасауға кімге рұқсат ететінін және қандай режимде (тек оқу, оқу және жазу, өшіру және т.б.) болуы қажет екенін көрсету керек. Кейбір жағдайларда операциялық жүйе пайдаланушыға берілген кілт бойынша деректерді шифрлеуді де қамтамасыз етті.

Жинақтауыштарда деректердің белгілі бір ұйымдастыруын (файлдық жүйе) қолдау операциялық жүйе құрамына жинақтауыштарды алдын – ала дайындауды орындайтын тағы бір арнайы бағдарламаны (утилита) қосуды талап етті. Сонымен бірге

жинақтауышқа пайдаланушы деректерін бұдан кейін де орналастыруды қамтамасыз ететін арнайы белгілеу жазылды.

Уақытты бөлу жүйесі. Жады көлемінің және өнімділігінің артуының келесі деңгейінде ірі есептеуіш құрылғы тек бір ұйымға ғана емес түгел бір ауданға қызмет көрсете алатыны түсінікті болды. Әрине бұл кезде пайдаланушылардың жүйеге алыстан қол жетімділігін қамтамасыз ету мәселесі туындайды. Қол жетімділік мәселесі телетайп негізіндегі арзан терминалдық құрылғыларды пайдалану есебінен шешілді.

Терминал (пернетақта және басым шығаратын құрылғы) есептеуіш құрылғыға қарапайым телефондық кабель (екі өткізгіші бар сым) арқылы жалғанды. Уақытты бөлу операциялық жүйесі бірнеше жүздеген терминалдардың қатар қызмет көрсетуін (енгізу/шығару) қолдап үлгерді, өйткені телетайпта енгізу және шығару уақыты секундына бірнеше ондаған символдардан аспайды. Пайдаланушыда өзінің жеке есептеуіш машинасына қол жетімділігі бар деген иллюзиясы пайда болды. Оған енді оператор көмегінің қажеті болған жоқ.

Әрине процессор уақытын бөлу мульти бағдарламалаумен қамтамасыз ету деңгейінде қолданылды. Бірақ салыстырмалы түрде жылдамдығы аз құрылғылар арқылы көптеген пайдаланушылардың қосылуы машина жадын қолдану арқылы уақытты бөлу мүмкіндігіне әкелді. Пайдаланушы терминалдағы хабарламаны оқып, жауап енгізгенше оның бағдарламасы тұрып қалды. Сол себептен күту режимінде тұрған бағдарламаны операциялық жүйе сыртқы жинақтауышқа (магниттік диск немесе магниттік барабан) ысырып тастап отырды. Пайдаланушыдан ары қарай жұмыс жасауға ақпарат түскен кезде бағдарлама оперативті жүйеге қайта қалпына келеді. Жұмыс режимінің мұндай түрі свопинг (ағылшын тілінен *swop* — орнын өзгерту) атауын алды.

Уақытты бөлу жүйесінің пайда болуына операциялық жүйелердің сыртқы құрылғылармен алмасу қызметін өз мойнына алуы ықпал етті, және пайдаланушы терминалын кез-келген пәтерден жалғауға мүмкіндік беретін пайдалануда айтарлықтай арзан телефон желілері пайда болуы оң әсер етті. Әртүрлі пайдаланушылардың жеке мәліметтерін дербестендіру алдыңғы деңгейде шешілді.

Механикалық телетайптардың сенімділігі жоғары деңгейде болған жоқ, сол себепті электронды-сәулелік түтік негізіндегі терминалдар тез пайда болды, олар шығарудың жоғары жылдамдығына ие болды және сенімділік тұрғысынан көрсеткіштері жоғары болды. Әдетте оларды мәліметтердің қатты көшірмелерін алу үшін шағын басып шығару

құрылғылары (принтерлер) толықтырып отырды.

Терминалдардың осындай конфигурациясы Олимпиада-80 ақпараттық кызмет көрсету үшін Мәскеуде 1980 ж. және 1985 ж. Жастар және студенттер фестивальін өткізгенде кеңінен қолданылды. CDC фирмасының оқу терминалдары құрылған, оларды электронды-сәулелік түтіктің орнына плазмалы панель орнатылған. Бұл символдардың бейнеленуінің жоғары тұрақтылығын қамтамасыз етті, оған қоса, экран жазықтығы оған микрофильмдер суреттерін проекциялау мүмкіндігін берді, осылайша байланыс жолағындағы мәтіндік ақпаратты толықтырды.

Терминалға мәліметтерді алдын ала өңдеуді тапсыруға да болатыны өте тез түсінікті болды: символдық жолақтарды редакциялау, шағын көлемді деректерді сақтау, баспаны басқару. Бұндай терминал байланыс желісінен де орталық есептеуіш құрылғының да жүктемесін жеңілдетеді. Интеллектуалдық терминал туралы айтыла бастады.

Дербес компьютерлер және желілер. Келесі қадам интеллектуалдық терминалды өзінің орталықтандырылған процессоры бар, өзінің оперативті жады және сыртқы құрылғылар жиынтығы бар өз бетімен есептеу құрылғысына айналдыру. Тіке қолжетімді магниттік жинақтаушы негізінде деректерді жинау шағын құрылғысының тез дамуы олардың сыйымдылығы және қолжетімділік жылдамдығы пайдаланушының жеке ақпараттарын орналастыруға толық жететін дәрежеге жетті. Бір сәтте пайдаланушылардың барлық мәселелері шешілгендей болып көрінді. Олар монополиялық иемденуге өздерінің жеке есептеуіштеріне ие болды — дербес компьютер (ДК, Personal Computer — PC).

ДК бірінші операциялық жүйелерінде мультибағдарламалану дамыған құрылғылары және деректерді сақтау құралдары болған жоқ. Бұл бір пайдаланушылы, бір мақсатты операциялық жүйелер болды, бірақ бұлай ұзаққа жалғасқан жоқ. Орталықтандырылған деректер қорына қолжетімділік алуға деген талпыныс байланыс желілердің дамуына алып келді. Пайдаланушының тапсырмаларын орындауда және әртүрлі мәселелер қатарында тек бір ДК ресурстарымен қамтамасыз етіле алмады. Жады көлемінің өсуі және ДК микропроцессорлардың тез әрекет етуі мульти бағдарламалау және көп мақсаттылыққа өтуге мүмкіндік берді.

Алдыңғы деңгейде дамыған телефон желілері деректердің қажет көлемін жіберуге шамалары келмеді. Көлемі жүз байт жолақтық ақпаратты орталық есептеуіштен терминалға жіберу кезінде жеткілікті болған секундына мың және он мыңдаған биттер жылдамдығы жеткіліксіз болды. Шынымен де, 1 Мбайт қарапайым суретті жолдау

үшін бір жарым-екі минут қажет болады.

Сол себептен ДК дамуы, олардың пайда болу бірінші деңгейіне қаншалықты қызық болып көрінгенмен жоғары жылдамдықты байланыс желілерінің дамуын лезде ынталандырды. Операциялық жүйелер құрамында желілік алмасу бағдарламалық қызметінің болуы ажырамас бөлікке айналды. Негізінде бұл атқарымдарды стандартты атқарымдар кітапханасының ары қарай дамуы деп қарастыруға болады. бір жағынан қарасақ — бұл желілік қолжетімділік ережелерін және ұйымдастыру туралы өз келісімдері бар деректерді басқарудың заманауи жүйесінің бөлігі.

1.4.

UNIX және WINDOWS ОПЕРАЦИЯЛЫҚ ЖҮЙЕЛЕР ТЕГІ

Windows және UNIX тегінің операциялық жүйелері және олардың тармақталуы қазіргі уақыттағы ең танымал және белгілі болып табылады. UNIX операциялық жүйелері операциялық жүйелердің ішіндегі ең ескі тармағы болып саналады, UNIX-операциялық жүйелері әлі де кеңінен қолданылады және өз өзектілігін жоғалтқан жоқ.

Бір операциялық жүйе туралы емес, UNIX тегінің операциялық жүйелері туралы айтудың басты себебі, UNIX -тегі әртүрлі, көп жағдайда бір-бірімен байланысты емес өнертапқыштар ұжымдарында ұзақ уақыт дамыды.

UNIX ОЖ алғашқы нұсқасы 1969 ж. AT&T фирмасының Bell Labs зертханасының бірнеше бағдарламашылары шығарды және PDP-7 компьютерінде жұмыс жасады. Операциялық жүйе зертхана қызметкерлерінің тәжірибелік тапсырма міндеттерін шешу мақсатында пайдаланылды және де оны кеңінен тарату жоспарда болған жоқ. Біршама уақыт өткеннен кейін операциялық жүйенің көп бөлігі ассемблер тілінен C тіліне қайта жазылды, бұл өз кезегінде операциялық жүйені басқа көптеген платформаларға ауыстыруға мүмкіндік берді. Қазіргі таңда UNIX көптеген ағымдағы архитектураларға жұмыс жасайды және көбісіне басты ОЖ болып табылады.

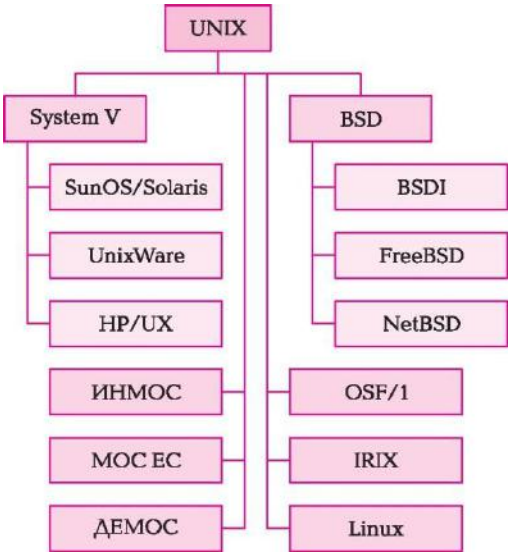
AT&T ойлап шығарылған UNIX-жүйелерінің ары қарай дамуы және олардың туындысы System V (бесінші нұсқа) деп аталады, қысқартылуы SysV (кейде AT&T-нұсқасы UNIX деген атау

қолданылады).

1970 ж. ортасында UNIX бастапқы код Беркли университетіне тап болды, ол жерде UNIX жаңа нұсқасы пайда болып, BSD UNIX (Berkeley Software Distribution) атау алды.

Қазіргі кезде UNIX көптеген нұсқалары System V немесе BSD (сурет - 1.3) негізделген.

Екі тармағы да әртүрлі стандарттарды әр алуан дәрежеде қанағаттандырады, олардың ішінде POSIX стандарты бар. Соңғы кезде бірдей стандарттар жасалу үстінде. Осы стандарттардың талаптарын



Сурет - 1.3. UNIX әртүрлі нұсқаларының генеалогиялық тармақ

әзірлеу Silicon Graphics), Digital OSF/1 (DEC әзірлеуі) және Энди Таненбаум ойлап шығарған MINIX негізделген ескі нұсқа Linux.

Одан бөлек, UNIX тобының ОЖ генеалогиялық тармағында отандық талдамасы жатқызуға болады: ЕС ЭВМ арналған операциялық жүйелер МОС ЕС, ЕС ЭВМ және СМ ЭВМ арналған ДЕМОС, СМ ЭВМ арналған ИНМОС [5].

Windows тобының операциялық жүйелері туралы сөз ететін болсақ мүлдем басқа көрініс орын алады. Бұл операциялық жүйелер 20 жылдан көп уақытта жалғыз бір компания ойлап шығарды және әлі де шығару үстінде.

Windows операциялық жүйелер тарихы өз бастауын 1985 ж. алады, сол кезде Windows 1.0 ОЖ ойлап табылды. Бұл жүйе толыққанды ОЖ болған жоқ, көп бөлігі MS DOS операциялық жүйесі үшін графикалық қабаты ғана болды. Ол көпмақсатты режимнің шектелген қолдауын ОС MS DOS бағдарламаларына ыңғайлап жүзеге асырды. Дәл осы уақытта Microsoft IBM фирмасымен бірге толыққанды, көпмақсатты, графикалық интерфейсі — OS/2 операциялық жүйені ойлап шығарды. Алғашқыда, дәл осы OS/2 операциялық жүйе ескірген MS DOS алмастыруға бағытталды, бірақ 1990 жылдардың басында Microsoft және IBM жолдары екіге бөлініп, Microsoft жеке өзінің Windows

негізіндегі операциялық жүйесін дамытуға бар күш-жігерін салды.

1990 ж. Windows операциялық жүйелер дамуының елеулі қадамы болған Windows 3.0 ОЖ жарыққа шықты. 1992 ж. Windows 3.x бұтағының айтарлықтай қайта өңделген жаңа нұсқасы шықты — Windows 3.1. ол нұсқада TrueType әріптерін пайдалану мүмкін болды, WYSIWYG режимінде баспа, жұмыс тұрақтылығы артты, мультимедиялық қасиеті және т.б. қосылды.

Windows операциялық жүйесі тармағының келесі даму деңгейі Windows 9x ұрпағы болып табылады. Бұл топқа операциялық жүйенің үш нұсқасы кіреді: Windows 95, Windows 98 және Windows ME. Бұл операциялық жүйелер жаңартылған пайдаланушылық интерфейсті, файлдардың ұзақ атауын қолдауды, 32-биттік қосымшаларды қолдауды, TCP/IP стек қолдаудан асып түсті. Осы операциялық жүйелер жұмысының тұрақтылығына байланысты айтарлықтай үлкен кейітулер шақырды. Бұл операциялық жүйелердің нақты ядросы бір-бірінен процестерді оқшаулауға мүмкіндік беретін ығыстыратын көп мақсаттылықты (свопинг) қолдайды және бір процестің тежелуі әсерінен барлық жүйенің тоқтатылуының алдын алады. Ядро бөлігінің орындалуы ОС Windows 3.x ядросынан ерекшеленбегендіктен, мысалы жүйедегі жалғыз 16-биттік қосымшаның қатып қалуы барлық операциялық жүйенің қатып қалуына әкеліп отырды.

Windows NT (New Technology) тармағы негізінде жүйелер мүлдем басқа қағидамен жасап шығарылды. Осы тармақтың Windows тобының операциялық жүйелерін шығару 1988 ж. басталды. Бұл ұрпағы Windows операциялық жүйелерінің дамуындағы ең сәтті тармақ болды.

Windows NT 3.1 бірінші нұсқасы 1993 ж. шықты. Содан кейін арасына жыл салып бірінен кейін бірі Windows NT 3.5, 3.51, 4.0 нұсқалары жарыққа шығып отырды. Windows NT 4.0 нұсқасының интерфейсі Windows 95 стилінде болды, бірақ DirectX — ойын қосымшаларын қолдауға арналған кітапхана қолдануының болмауынан зардап шекті.

2000 ж. Windows NT ядросы негізінде жаңа Windows 2000 операциялық жүйесі шықты. Жүйе жаңа пайдаланушылық интерфейсті қосты, NTFS 3.0 файлдық жүйесін қолдады, сонымен қатар құрамында айтарлықтай көп өзгерістер болды.

2001 ж. Windows XP операциялық жүйе пайда болды. Бұл нұсқа Windows 2000 операциялық жүйелер өзгертуге арналған, сонымен қатар, сол уақытта зерттемесі тоқтатылған Windows 9x бұтағының операциялық жүйесінің ауыстыру ретінде ұсынылды. Бұл операциялық жүйе жаңа графикалық интерфейске, барлық бар болуы мүмкін

мультимедиалық атқарымдардың кең қолдауына, жүйені қайта қалпына келтіру қызметіне, көне бағдарламалар мен ойындармен жақсартылған үйлесімділікке ие болды.

2006 ж. Windows Vista операциялық жүйесі жарыққа шықты. Бұл жүйе DirectX негізінде жаңа пайдаланушы интерфейсін Windows Aero қолдады, User Control (UAC) жүйесі пайда болды. Іс-әрекеттерді бақылау Data Execution Prevention және Address Space Layout Randomization (ASLR) және т.б. эксплойттарын қолданудың алдын алу технологиясы қолданылды.

Көптеген жаңадан енгізілулерге қарамастан бұл нұсқа кең талқыға түсті және әлем бойынша кең қолданысқа ие болған жоқ. Осы нұсқаны алмастыру ретінде 2009 жылы Windows 7 операциялық жүйесі шықты. Бұл жүйе жаңа DirectX 11-нұсқаға ие болды, ескі қосымшалармен үйлесімділігі артты (Windows Vista салыстырғанда), жүйенің пайдаланушылық интерфейсі өзгеріске ұшырады және т.б.

2012 ж. Microsoft фирмасы Windows 8 операциялық жүйесін шығаруға дайындады. Бұл жүйе құрамындағы ерекшеліктер дауысты тану механизмдерінде қосымша жақсартулар бар, зиянды бағдарламалардың енуінен қорғайтын жақсартылған жүйе, интерфейсі өзгерген, Windows Phone 7 мобильдік нұсқасының интерфейсіне ұқсайды, және т.б.

Операциялық жүйелердің пайдаланушыларға арналған нұсқасынан бөлек Windows топтары серверлерге, деректерді өңдеудің үлкен орталықтарына арналған арнайы нұсқалары бар. Пайдаланушыларға арналған әр жаңа Windows нұсқасы шыққанда, әсіресе Windows NT ядросы негізінде, операциялық жүйенің серверлік нұсқасы қатар шығады (іс жүзінде сол ядрода). Осылайша келесідей серверлік нұсқалар шықты, Windows Server 2003, Windows Server 2008, Windows Small Business 2008, Windows Server 2008 R2, Windows Server 8.

1.5.

«БІЛІМДІ БАҚЫЛАУ» МІНДЕТІНІҢ ҚОЙЫЛУЫ

Материалды түсіндіруді түсінікті ету мақсатында оқулықта келтірілген мысалдардың басым бөлігі қосымшаны біртіндеп жасап шығаруға арналады. Ол қосымша студенттерге бақылау жұмысын тарату және орындалған жұмысты оқытушыға қайтадан жинап беруді автоматтандырады. Қосымшаның шартты атауы — «Білімді тексеру».

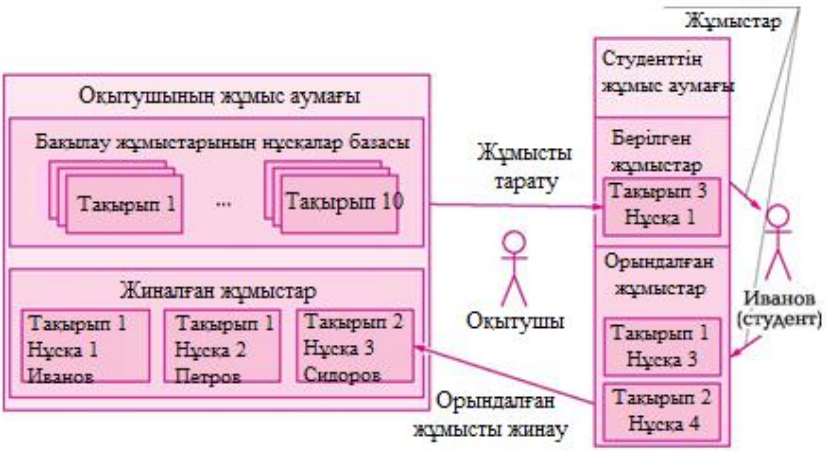
Сонымен қоса оқытушы мен студенттер арасындағы қарым-қатынастың кейбір аспекттері материалды баяндауды қысқарту үшін әдейі жеңілдетіледі. Операциялық жүйе ұсынатын әртүрлі механизмдердің жұмысы мен қолданылуын көрсететін сәттер ғана ерекшеленеді. Бұл қосымшаға арналған жиынтық мысал 1-Қосымшада келтірілген.

Оқулықтың негізгі бөлімінде «Білімді тексеру» қосымшасын сипаттауға қатысты таңбасыз әріппен ерекшеленген және негізгі мәтінмен салыстырғанда сол жақтан үлкен бос жерден басталады (келесі абзацтағыдай).

«Білімді тексеру» қосымшасы оқытушының студенттерге бақылау жұмыстарын тарату процесін және орындалған жұмыстарды қайтарып алу процесін автоматтандырады (сурет-1.4).

Бұл қосымша негізгі пайдаланушының үш түріне арналған:

- қолданбалы бағдарламалаушы — қолданбалы жүйе зерттемесін орындайды, оның қызметін кеңейтеді;
- оқытушы — әртүрлі тақырыптар бойынша бақылау жұмысы нұсқаларының базасын қолдайды, студенттерге жұмыс нұсқаларын таратады, жазылған жұмыстарды жинайды да тексереді. Жұмыстарды талдау және баға қою жүйе шеңберінен шығарылған;
- студент — бақылау жұмысының алынған нұсқалар тізімін қарайды, оларды орындайды да, орындалған жұмысты оқытушыға қайтарады.



Сурет - 1.4. «Білімді тексеру» жүйесінің жұмыс жасау сұлбасы

«Білімді тексеру» Қосымшасы тақырып бойынша топтастырылған бақылау жұмыстардың нұсқалар базасын сақтауға арналған құралды ұсынуы қажет. Әр тақырыптың бірегей номері болуы керек, әр нұсқаның —тақырып айналасында бірегей болатын номері болады.

Студенттердің жұмысы үшін оқытушы орындауға арналған жұмыс нұсқаларын орналастыратын жұмыс алаңы бөлінеді.

Студенттің жұмыс алаңының ішінде арнайы бос алаң болуы қажет, ол жерге студент орындалған жұмыстарды салады. Дәл осы арнайы алаңнан оқытушы жұмыстарды тексеру үшін алады да өз жұмыс алаңына орналастырады.

Осылайша, «Білімді тексеру» қосымшасында сақталған негізгі объекттерді анықтауға және жүйе пайдаланушылары ол қосымшамен жүзеге асыратын әрекеттерді білуге болады:

- Бақылау жұмысының *база*сы — қолжетімді нұсқалар туралы ақпараттың негізгі қоймасы. Деректердің негізгі объекті — нұсқа. Нұсқалар тақырып бойынша топтастырылған, тақырыптар ортақ базаны құрайды;
- студенттің жұмыс орны — студентке берілген бақылау жұмысы нұсқасын сақтау қоймасы. Әр студентте өзінің жеке жұмыс орыны бар.
- Деректердің негізгі нысаны — бақылау жұмысының нұсқасы;
- студенттің дайын жұмыс аймағы — студенттер орындаған, тексеруге дайын бақылау жұмысын сақтайтын орын;
- мұғалімнің жұмыс аймағы — студенттерден жиналған бақылау жұмыстарын жинайтын орын;
- бақылау жұмысының нұсқасы — сұрақтар мен жолақтар тізімі, студент жауаптарын жазуға арналған;
- бақылау жұмысының орындалған нұсқасы — студент жауап үшін толтырған бақылау жұмысының нұсқасы.

Жүйе пайдаланушының негізгі әрекеттерін автоматтандыруы қажет.

Мұғалімнің жұмысы:

- белгілі бір тақырып бойынша нұсқалар санын тексеру;
- нақты бір студентке тақырып бойынша бір нұсқа беру;
- барлық студенттерге берілген тақырып бойынша нұсқаларын

тарату;
- орындалған жұмыстарды өз жұмыс орнына жинау.

Студенттің міндеті:
- бақылау жұмысында берілген нұсқаларды қарау;
- бақылау жұмысын орындау;
- бақылау жұмысын оқытушыға тапсыру.

«Білімді тексеру» қосымшасын жазған кезде келесі мәселелер ескерілуі қажет:

- ақпаратты дискті жинақтауышта орналастыру және құрылымдау;
- әртүрлі ақпаратты қолжетімділік құқығын анықтау;
- пайдаланушының әдеттегі әрекеттерін автоматтандыру құралдары.

Интернет-дереккөздері

https://secшewikimedm.org/wikipedm/ra/wiki/Нақты_уақыттағы_операциялық_жүйе
http://wwwrtsoft-trammg.ra/?p=600067&PHPSESSro=9f9bebc48282ab1cc30cd35794ccb576
https://secure.wikimedia.org/wikipedia/en/wiki/Real-time_operating_system
http://www.osp.ru/os/1997/05/179265/

БАҚЫЛАУ СҰРАҚТАРЫ

1. Операциялық жүйе қандай негізгі қызметтерді атқарады?
2. Көп пайдаланушылы және бір пайдаланушылы ОЖ ядроларының не айырмашылықтары бар?
3. Ядро құрамына не себептен кіріс/шығыс қызметі енеді?
4. Өз құрамында файлдық жүйе болмайтын ОЖ бар ма?
5. Қолданбалы бағдарламашы мен қолданбалы пайдаланушының операциялық жүйемен байланыс құралдары бір-бірінен қалай ерекшеленеді?
6. Операциялық жүйенің қандай қызметтері қолданбалы бағдарламаның деректерді жинақтауыштарда физикалық ұйымдастырудан тәуелсіз болуын қамтамасыз етеді?
7. Ядросында файлдық жүйені қолдайтын қызметі жоқ ОЖ болуы мүмкін бе?

Тарау 2

ФАЙЛДЫҚ ЖҮЙЕЛЕР

2.1.

ДИСКТЕ АҚПАРАТТЫ САҚТАУДЫ ҰЙЫМДАСТЫРУ

Жүйемен жұмыс сеансы барысында пайдаланушы өзінің деректерін жаңартады және өзгертеді, мысалы құжаттар, суреттер, бағдарламалар мәтіндері. Көп жағдайда пайдаланушы мәтін теріп, оның қатты көшірмесін шығару мақсатында баспадан шығарып өз жұмыс сеансын аяқтай алады. Бұл кезде пайдаланушы өз жұмысының нәтижесін алады — басып шығарылған нұсқасы, ал терілген мәтін аралық дерек болып саналады, оны сақтаудың қажеті жоқ. Көп кездесетін жағдайларда мәтін көлемі көп болып, пайдаланушы оны бір жұмыс сеансында аяқтай алмауы ықтимал. Ондай жағдай болса оны пайдаланушы жүйеге жүгінгенде сақтау қажет. Мәтін сақталатын дерекке айналады.

Пайдаланушы деректерін ұзақ сақтау үшін деректерді жинақтауыштар қолданылады. Олар — дискті (қатты дисктер, CD- және DVD-дисктер, флоппи-дисктер), ленталық (стримерлер) немесе қатты денелі (flash-жинақтауыштар). Жинақтауышпен байланысты операциялық жүйе жасайды, ал пайдаланушы деректерінің бірлігі бұл жерде файл болып саналады.

Файл (кейде оны деректер жинағы деп атайды) пайдаланушы белгілейтін бірегей атауы бар, дискте сақталатын деректер деп қарастыруға болады. Файл деректерінің пайдаланушының қолданбалы бағдарламалары қолдануына арналған форматтары болады. Мысалы, кітап дискте book.txt атауы бар файл түрінде сақтала алады, ал мәтіндік форматта txt.

Файлдар атауы ұзақтығы шектеулі символдық жолақтар (әдетте 8 дейін немесе 255 символға дейін), олар файлда сақталған деректердің идентификациясы үшін қызмет етеді. Ереже бойынша файлдар атауында қолдануға тыйым салынған таңбалар қатары бар. Мысалы, UNIX-жүйелерде *, ?, /, \, <, >, &, $, | және басқа да символдар қатары.

Осы формат файлымен жұмыс істейтін бағдарламаны салыстыруды жеңілдету үшін оның түрін беруге болады. Ереже бойынша файл типі кеңейтілу көмегімен беріледі, нүктемен бөлінген файл атауының бөлігі.

Осылайша book.txt атауының ішінде, book — атауы, ал txt — кеңейтілуі. Файлдың ішкі құрылымы кеңейтілуден тәуелді емес, ал файлдар типтендіру Windows жүйелерінде жасалғандай тек кеңейтілу көмегімен ғана жүргізілмейді. Мысалы, UNIX-жүйелерінде бағдарламалардың орындалу үстіндегі файлдары әдетте кеңейтілуі болмайды, оның орнына оларға «орындалу үстінде» арнайы атрибут меншіктеледі.

Файл атауында типін көрсету оның өңделу форматын анықтамайды. Файлмен жұмыс істейтін бағдарламаға байланысты мәліметтер мүлдем әртүрлі болуы мүмкін. Мысалы, мәтіндік редакторда ашық (txt кеңейтілуі бар) мәтіндік файл символдар кезегін білдіреді – мәтін, осындай жағдайда орфографияны тексеру үшін осы мәтінді құрайтын бөлек сөздер қажет. Резервтік көшіру бағдарламасы үшін дәл осы файл 16 Кбайт тұратын деректердің кезекті блогын білдіреді. Осылайша, файл құрылымы мен онымен жұмыс жасау тәртібін бағдарлама өзі енгізеді.

2.2.

ФАЙЛДЫҚ ЖҮЙЕЛЕР

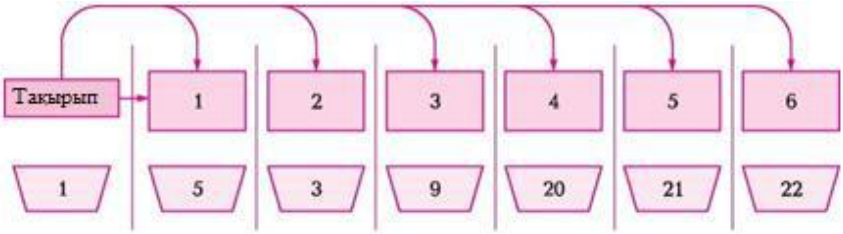
Операциялық жүйе тарапынан файл дегеніміз не? Дәлірек тоқталсақ, файлдық жүйе тұрғысынан ОЖ ядросының бөлігі ретінде файл дегеніміз не?

Операциялық жүйе қолданбалы бағдарламаларға файлдарға қолжетімділік интерфейсін ұсынады, бірақ өздігімен файл қандай ақпараттан тұратынын қарайды. Автокөлік үлгісін жасап шығаруға болатын ойыншық құрастырғышты көз алдымызға елестетейік. Дәл осылай операциялық жүйе бөлек «бөлшектерден» — блоктардан файлдарды жинайды.

Осы ретте дәл құрастыру сұлбасы ішіне бірге салынған құрастырғыштағыдай ОЖ бөлек блоктарды құрастыру ережесін басшылыққа алады. Бұндай құрастырылған блоктардың жинағы деректер жинағы деген атауға ие болды. Деректер жинағы — бұл файлға қарағанда деректердің әлдеқайда төмен деңгейлі ұғымы. Оның ұйымдастырылуы көп жағдайда деректер жинақтауыштың қасиеттерімен анықталады.

Деректер жинағының файл секілді бірегей атауы бар, файл құрамында бар барлық ақпаратқа ие, бірақ бұл ақпараттардың құрылымы операциялық жүйенің ядросымен анықталады. Онымен қоса, деректер жинағына басқа бағдарламаларға қолжетімді емес. Қызметтік ақпарат

кіруі ықтимал.

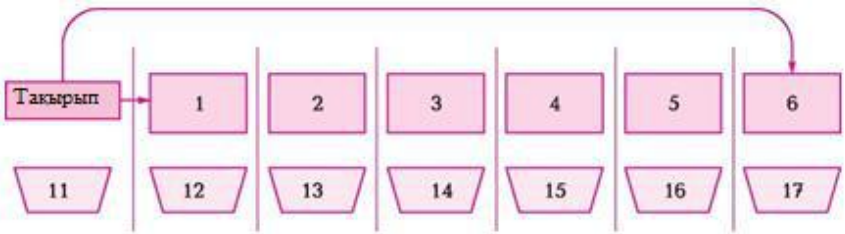


Сурет-2.1. деректер жинағының барлық блоктарына сілтемелері бар файлдық жүйелер

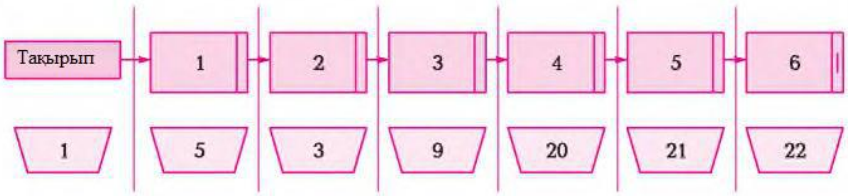
Файлдық жүйе қолданатын барлық дискті кеңістік бөлек блоктарға, яғни кластерлерге бөлінеді (әдетте қолданатын көлемдері 1, 2, 4, 8 немесе 16 Кбайт). Әр кластердің өзінің номері бар, пайдаланушы ақпаратын немесе қызметтік ақпаратты сақтайды. Бұл қызметтік ақпарат деректер жинағына блоктарды жинау үшін де қолданылады. Кластер өлшемі файлдық жүйені жасаған кезде бекітіледі. Мысалы, қызметтік блок деректер жинағына кіретін блоктар номерінің кезегін сақтай алады (сурет – 2.1).

Деректерді осылай ұйымдастыру тәсілінің кемшілігі үлкен файлдарда блоктар номерлерінің тізімі бір кластерден көп кластерге орналасуы мүмкін. Нәтижесінде файл көлемі шектелген болып шығады. UNIX-жүйелерде осы шектеуді айналып өтудің бірнеше тәсілі бар, мысалы қызметтік блоктарды *дарақ*қа ұйымдастыруға болады, сонымен қатар әр қызметтік блок деректері бар блоктер кезегін және тармақтың келесі деңгейінің қызметтік блоктарын сақтайды.

Деректер блогын ұйымдастырудың басқа тәсілі келесідей: деректер жинағы кезекпен тұрған кластерлерге орналасады, бұл жағдайда қызметтік блокта бірінші және соңғы кластерлер номерлерін сақтаған жеткілікті (сурет -2.2).



Сурет-2.2. Деректер блогын кезекпен ұйымдастыру файлдық жүйесі



Сурет - 2.3. Деректер блоктарын тізім бойынша ұйымдастыру файлдық жүйесі

Деректердің жиі өзгеруі кезінде ұйымдастыру тәсілі қолайлы емес. Деректер жинағының көмегі ұлғайған кезде соңғы блоктан кейін қажет көлемде бос блоктар болуы керек немесе деректер жинағын әр сайын бір орынға орналастырып отыру қажет. Дегенмен деректерді ұйымдастырудың негізіне ұқсас қағидалары CD-ROM-жинақтауыштарға салынған, ал бұған дейін ОС РАФОС қолданылды.

Ұйымдастырудың тағы бір тәсілі әр блокта шағын қызметтік орын бөліп, сол жерге деректер жинағының келесі блок номерін жазып сақтау (сурет–2.3). Осылайша, деректер жинағы сызықтық тізімде ұйымдастырылады, ал қызметтік блокта пайдаланушы дерегімен бірге бірінші блоктың номерін сақтаған жеткілікті.

Жинақтауыштағы деректер жинағы құрылымы туралы келісім файлдық жүйенің бөлшегі болып саналады, ол үш құрауыштан тұрады:

1) Деректерді жинақтауышта сақтау құрылымы туралы келісім — жинақтауышқа орналастырылуы мүмкін ақпарат типін анықтау (мысалы пайдаланушының/қызметтік ақпарат), сонымен қатар жинақтауышта деректер орналасуына қатысты ережелер жинағы;

2) Деректерді өңдеу ережелері туралы келісім — ережелер жинағы осы ережелерді негізге ала отырып деректер өңделеді, мысалы «файл, оған деректер жазылмай тұрып ашық болуы тиіс»;

3) ОЖ ядросының құрамына кіретін деректерді өңдеу процесі де жоғарыда көрсетілген келісімдерге бағынады.

Сондықтан файлдық жүйе туралы айтқан кезде оның қай аспектісі туралы сөз қозғалып отырғанын жиі анықтау керек.

ОЖ арналған деректер жинағы идентификаторы ретінде әдетте қызметтік блок кластері алынады. Операциялық жүйе деректер жинағы идентификаторы мен файлдар атауы арасында сәйкестікті қолдайды. Бұл пайдаланушыдан деректерге жүгіну механизмін

жасыруға мүмкіндік береді — пайдаланушы файлды атауы арқылы іздейді, ал операциялық жүйе осы атау бойынша деректер жинағы идентификаторын табады.

Деректер жинағы идентификаторы мен файлдар атауы арасында сәйкестік файлдарды орналастыру кестесі деп аталатын дисктік кеңістіктің арнайы аумағында сақталады. Әр деректер жинағына файлдардың бірнеше атауы салыстырылуы мүмкін. Осындай әр сәйкестік файлдарды орналастыру кестесінде бөлек жазба болып жазылады және UNIX-жүйесінде қатты сілтеме деп аталады.

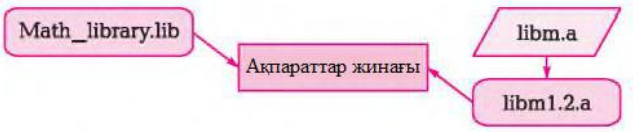
Қатты сілтеме бір дерекке бірнеше файлдар атауымен жүгінуге жағдай жасайды. Бұл деректі әрқайсысы тек белгілі бір кеңейтілуі бар файлдармен ғана жұмыс істейтін бірнеше бағдарлама көмегімен өңдеу қажет болса ыңғайлы.

Деректер жинағы қызметтік блогының UNIX-жүйесінде деректерге қолжетімділік режимі туралы ақпарат (қолжетімділік құқығы, 6.4 тарауды қарау қажет) сақталатындықтан, бұл ақпарат барлық қатты сілтемелер үшін бірдей болады.

Қатты сілтемелерден бөлек символикалық сілтемелер бар — осы сілтеме көрсететін файл атауын сақтайтын арнайы файлдар (сурет – 2.4). Осылайша символикалық сілтеме деректер жинағына емес файлға сілтеме жасайды. Бұл сілтеме жиі өзгертілгенде немесе әртүрлі сілтемелерге әр түрлі қолжетімділік құқығын беру қажет болған кезде ыңғайлы.

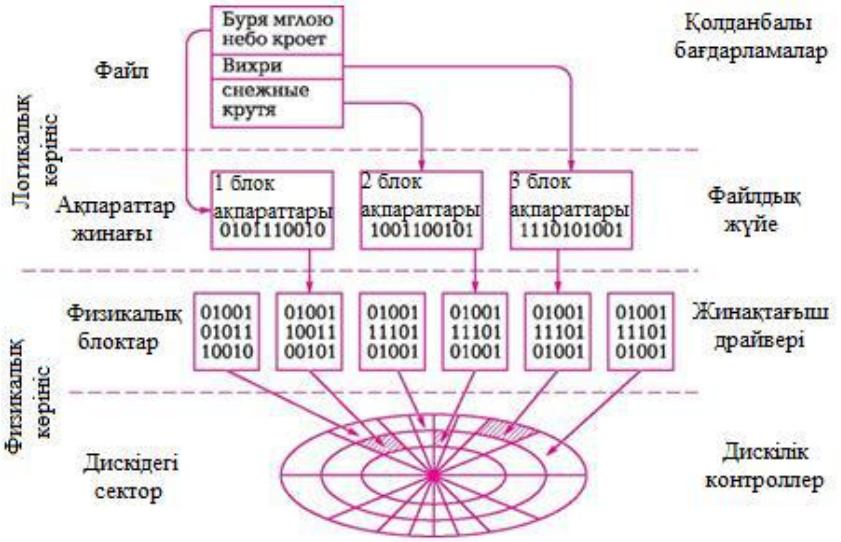
Деректер жинағының құрылымы көп жағдайда деректерге қолжетімділік режимін анықтайды. Мысалы, деректер жинағының блогын тізбекті кластерлерге орналастырған кезде немесе сызықтық тізім пішінінде ұйымдастырған кезде деректерге тізбектей қолжетімділікті ұйымдастырған оңай — берілген деректер орнына барлық жүгіну тек алдыңғы деректер блогының барлығын алдын ала асып кеткеннен кейін мүмкін. Блоктар номері тізімін сақтауды ұйымдастыру туынды қолжетімділілікті ұйымдастыру кезінде өте ыңғайлы — белгілі бір блокты алу үшін оның номері қызметтік блоктан тауып алған деп есептеу жеткілікті.

Деректерге қолжетімділік режимі тек файлдық жүйемен анықталмайды. Егер деректер құрылымын және оның ары қарай өзгеруін жинақтауыштағы деректердің физикалық құрылымына дейін қарай беруді жалғастырсақ, деректер бірнеше түрлендіруден өтетінін көруге болады (сурет – 2.5), сонымен қатар әр деректерді пайдаланушы (мысалы, пайдаланушы бағдарламасы немесе файлдық жүйе) өздерінің



деректеріне қолжетімділік режиміне өз шектеулерін қояды.

Сурет- 2.4. Деректер жинағы, қатты және символикалық сілтемелер



2.5. сурет мәліметтерді ұсыну деңгейлері

Жоғарыда көрсетілген деректерді ұсыну деректердің логикалық құрылымын қалыптастырады — бағдарламалық қамтылыммен пайдалануға ыңғайлы құрылым, жинақтауыштағы деректердің физикалық сипатталуымен ешқандай ұқсастығы жоқ. Егер деректердің физикалық құрылымын қарастыратын болсақ, ең төменгі деңгейде деректер дисктің минималды адрестелетін аумағы, яғни магниттік домен (қатты немесе иілгіш дискте) ретінде ұсынылған. Бір магниттік домен дегеніміз бір бит ақпарат, сектор көлемі әдетте 512 байт тең, дегенмен бұл мән дисктік бөлім жасаған кезде өзгеруі мүмкін. Көлемі 512 бастап 4 096 байт дейін диапазонында ауытқиды. Сектор санауыш бастиек номері көмегімен, жолақ номерімен және жолдағы сектор номерімен сәйкестендіреді. Секторды есептеу немесе жазу кезінде ОЖ ядросына кіретін осы дисктік жинақтауыш драйверімен басқарылатын дисктік бақылағыш арқылы өтеді.

Дисктік бақылағышы бар драйвер интерфейсі құрылғыға қолжетімділік режимін анықтайды: символдық қолжетімділік кезінде ақпарат біртіндеп бір таңбамен есептеледі, блоктық қолжетімділік кезінде — белгіленген көлем блоктарымен (әдетте сектор өлшеміне *еселі*).

Драйвер құрылымдамаған номерленген физикалық блоктар тізбегі түріндегі дисктік кеңістікті ұсынады. Физикалық блок көлемі әдетте

сектор көлеміне тең. Деректердің физикалық түрінен логикалыққа ауысқан кезде физикалық блоктар файлдық жүйе жұмыс істейтін кластерлерге біріктіріледі.

Осылайша, файлдармен жұмыстың барлық жүйесі негізгі төрт деңгейге бөлінген: жоғарғы екеуі деректердің логикалық ұсынысын анықтайды, ал екі төменгісі – физикалық түрлерін анықтайды.

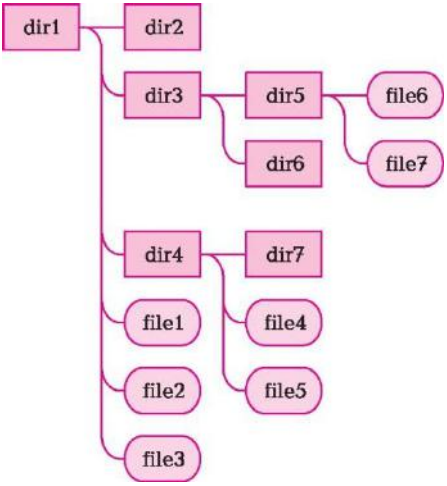
Қолданбалы бағдарламалаушының UNIX-жүйесіндегі негізгі жұмысы файлдық жүйе деңгейінде жүретіндіктен жоғарыда тізілген файлдық жүйенің барлық қызметтерін біріктіру қажет:

- Ақпарат жинақтауышында деректерді ұйымдастыру туралы келісімді анықтау;
- Деректерге қолжетімділік жолдары туралы келісімді анықтау (тізбекті және еркінше);
- Файлдардың атауы туралы келісімді анықтау;
- Деректердің логикалық құрылымы туралы келісімді анықтау;
- ОЖ ядросы құрамына кіретін және жоғарыда көрсетілген келісімдер бойынша жинақтауыштардағы мәліметтермен жұмыс жасауға арналған тәсілдер жинағын анықтау.

2.3. КАТАЛОГТАР

Файлдарды реттеу және ұйымдастыру үшін операциялық жүйелерде каталог түсінігі бар. Каталог файлдар туралы және басқа каталогтар туралы жазбалардан тұрады. Қандай да бір каталогта жазбалары бар файлдар мен каталогтар осы каталог ішінде орналасқан деп саналады. Бұл анықтаманың рекурсивтілігі каталогтар өзегі туралы, сонымен бірге файлдарды ұйымдастыру үшін қызмет ететін иерархиялық жүйе туралы айтуға мүмкіндік береді (сурет-2.6).

Суретте келтірілген атауы dir деп басталатын элементтер каталогтар, ал атауы file деп басталатын элементтер — файлдар. Егер бір каталог екінші каталогтың ішінде орналасатын болса сыртқы каталог бірінші каталогқа аталық деп аталады немесе *шағын каталог*. Ішінде орналасқан каталог ішкі каталог деп аталады. Мысалы, dir3 каталогы dir5 каталогы үшін аталық, ал каталог dir2 —dir1 каталогының ішкі каталогы. Аталық каталогы жоқ, иерархияның жоғары деңгейінде орналасқан каталог түбірлік каталог деп аталады.



Сурет-2.6. файлдарды каталог бойынша орналастыру мысалы

Каталогтарды заманауи операциялық жүйеде пайдалану үш факторға негізделген:

1) Каталогтар операциялық жүйенің файлды іздеуін жылдамдатады. *Дарақта* іздеу немесе бірдей шарттарды іздеу әдетте сызықты тізімге қарағанда тезірек жүреді;

2) Каталогтар файлдар атауларының бірегейлігінен кетуге рұқсат береді. Әр файлдың атауы бірегей болуы керек, бірақ бұл бірегейлік каталог шеңберінен тыс болуы керек;

3) Каталогтар жинақтауыштардағы файлдарды жіктейді. Әдетте бір каталогқа қандай да бір ортақ сипатпен біріктірілген файлдарды орналастырады, мысалы, кітап тараулары немесе операциялық жүйенің жүктелген файлдар.

Сақтау тұрғысынан қарағанда файлдық жүйедегі каталог — бұл өзінің ішінде сақталған файлдар мен каталогтар атаулары мен атрибуттары жазылған арнайы түрдегі файл. Қарапайым файлдардан ерекшелігі каталогтарға тізбектей қолжетімділік мүмкін емес — каталогтармен жұмыс файлдармен жұмысқа қарағанда басқаша ұйымдастырылған. Каталогпен жұмыс жасаған кезде біз олардың ішіндегі бөлек жазбалармен жұмыс жасаймыз, яғни осы каталогта орналасқан файлдар мен каталогтар туралы ақпараттардан тұратын ақпараттық блоктармен жұмыс.

Қарапайым операциялық жүйелер тек бір каталогпен жұмыс істейтін файлдық жүйелермен жұмыс жасай алады. Ол файлдар сақталатын түбірлік каталог. Анағұрлым күрделі операциялық жүйелер шектелмеген тіркеме каталогтар тармағымен жұмыс жасайды. Каталогтар бұтақтарының тіркеме тереңдігін шынайы шектеулігі

тіркеме бұтақтың төменгі деңгейінде орналасқан файлдардың атауларының ұзақтығымен анықталады.

Файлдың толық атауы—мәтіндік жолақ, арнайы таңба-бөлгіштер арқылы барлық каталогтар атауы көрсетілген, тамырлық каталогтан файлға дейінгі жолды көрсетеді. UNIX-жүйелерінде бөлгіш ретінде қиғаш сызық таңбасы қолданылады «/». Мысалы, file6 файлының толық атауы /dir1/dir3/dir5/file6 болады.

Бірінші қиғаш түбірлік каталогты білдіреді, яғни түбірлік каталогта file1 орналасқан файлдың толық атауы /file1 болады. Толық атаумен берілетін жол файлдың немесе каталогтың абсолюттік атауы деп аталады. Мұндай атау жолы қатыстық абсолюттік санау нүктесінен басталады—түбірлік каталогқа байланысты.

Қатыстық жол түсінігін анықтау үшін ағымдағы каталог түсінігін енгізу керек. Пайдаланушының жұмыс сеансының ақпараттық ортасына пайдаланушы қазір жұмыс жасап отырған каталог туралы ақпарат кіреді.

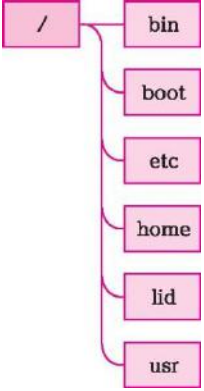
Ағымдағы каталогта орналасқан файлдар мен каталогтардың атаулары толық атауды бермей ақ, тікелей көрсетіле алады. Мысалы, егер ағымдағы каталог dir4 болса, онда file4 және file5 файлдарына атауларын толық көрсетпей-ақ, олардың тек өздерінің атауларын (file4 және file5) қолданып қана жүгінуге болады.

Қатыстық жол ағымдағы каталогқа қатысты анықталады, бірақ каталогтың өзінің атауы көрсетілмейді. Дегенмен де UNIX-жүйелерінде жолдың ағымдағы каталогтан екенін көрсету үшін мнемоникалық белгілеулер «.» жиі қолданылады. Егер ағымдағы каталог dir3 болса, онда оған қатысты file6 файлға апаратын қатыстық жол то ./dir5/file6 атаумен сұралады.

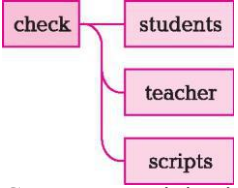
Аталық каталог жолын беру үшін «..» мнемоника қолданылады. Мысалы, егер ағымдағы каталог dir7, қатысты файл атауы file1 болса, жолы ../../file1 болады. алғашқы екі нүкте каталогты көрсетеді, dir7 үшін аталық, —dir4 каталогы, екінші екі нүкте — каталогқа, dir4 үшін аталық, —dir1 каталогы.

UNIX-ұқсас операциялық жүйелердегі жүйелік файлдар бөлек каталогтарда орналастырылған. Бұл әр түрлі файлдарды оларды пайдалануына қарай жіктеуге мүмкіндік береді. Көптеген UNIX-жүйелерінде жүйелік каталогтардың стандартты құрылымы бар (сурет-2.7).

UNIX-жүйелерінде ядроны (/boot), жүйелік кітапхана (/lib), жүйелік утилита кітапханасы (/bin), күйге келтіру утилитасы (/etc), пайдаланушылық бағдарламалар (/usr), пайдаланушылар деректерін (/home) сақтауға арналған бөлек каталогтар әдетте осылай



Сурет -2.7. UNIX-жүйесінің негізгі каталогтары



Сурет-2.8. «Білімді бақылау» жүйесінің негізгі каталогтары

ерекшеленеді. UNIX-жүйесінің стандартты каталогтар құрылымы туралы толықтай 7.1- бөлімде қарастырылған.

Біздің «Білімді бақылау» қолданыстағы тапсырмамызды шешу үшін деректерді ұйымдастыру құрылымын анықтау қажет. Ол үшін жүйелік файлдар UNIX қалай орналасса соған ұқсас жасауға болады — мәліметтер мен бағдарламаларды бөлек каталогтарға орналастырып, қосымша жіктеуді қажет ететін мәліметтерді деректер каталогының ішкі каталогына орналастырады.

«Білімді бақылау» қосымшасының бағдарламаларын және барлық деректерін сақтайтын каталог түбірлік каталогтың ішкі каталогы болсын (сурет-2.8), және атауын check деп алайық.

Қосымшаның деректері екі каталог бойынша жіктеледі — check каталогында студенттердің (students) және оқытушының (teacher) жұмыс үстелдерін сақтауға арналған каталог орналасады. Жүйедегі басты әрекеттерді орындауға арналған тапсырмалар scripts каталогына орналастырылады. Каталогтардың толық атаулары төмендегідей болады:

```
/check
/check/students
/check/teacher
/check/scripts
```

Әр студенттің жұмыс үстелі каталог болып табылады, оның атауы студенттің жүйедегі тіркелу атауымен сәйкес келеді. Әр студенттің каталогында орындалған жұмыстарға арналған бөлек ready ішкі каталогы бар.

Оқытушының жұмыс үстелінде әр тақырыпқа бір каталогтан арналады, тақырыптар нөмірленеді, ал каталогтар атаулары themel, theme2 және т.б. түрлері болады. Студенттерден жиналған жұмыстар үшін оқытушының жұмыс үстелінде арнайы бөлек каталог works болады.

2.4. ФАЙЛДАР МЕН КАТАЛОГТАРҒА ЖҮРГІЗІЛЕТІН ОПЕРАЦИЯЛАР

Файлдарға қолжетімділік үшін операциялық жүйе файлдармен жұмыстың негізгі қызметін жүзеге асыратын жүйелік шақыртулар жинағын ұсынады (кесте-2.1) [16].

Кесте- 2.1. Файлдармен жұмыс жасауға арналған жүйелік шақыртулар

Жүйелік шақырту	Сипаттама
Create	Шақырту кезінде ішінде деректер жоқ бос файл пайда болады. Шақырту файлға бірінше жазба жазылар алдында жасалады.
Delete	Файл атауын өшіреді. Егер деректер блогымен байланысты басқа файл атаулары жоқ болса, онда деректер блогы өшіріледі де, дисктік кеңістік босайды.
Open	Бұл шақырту кез-келген файлмен жұмыс басталар алдында жүргізілуі керек. Шақыртудың негізгі мақсаты — оперативті жадыға файл параметрлерін *оқу*, файлға кіріс-шығыс дисктік буфер бөлігін алып қою, және файлды ашқан бағдарлама жұмысы барысында файлға уақытша пайдаланушы идентификаторын тағайындау.
Close	Кіріс-шығыс буферіндегі, файл деректер жинағын сақтайтын дисктегі мәліметтерді жояды, буфер ресурстарын босатады, уақытша пайдаланушы идентификаторын өшіреді.
Read	Файлдан жалпы аумағына берілген байттар санын оқуға арналған. Оқу файлдағы ағымдағы позициядан басталады. Оқытылғаннан кейін ағымдағы позиция оқылған деректер соңынан баратын байтқа *көшіріледі.*
Write	Ағымдағы позициядан бастап мәліметтерді файлға жазуға арналған. Егер ағымдағы позиция файл соңымен сәйкес келсе деректер жинағы артады. Егер ағымдағы позиция файл ортасында орналасса, онда бар деректер жоғалады.

Жүйелік шақырту	Сипаттама
Append	Бұл жүйелілік шақырту Write шақыртуының қысқартылған пішінін білдіреді. Оның негізгі қызметі — файл соңына деректерді қосу.
Seek	Файлдағы ағымдағы позицияны белгіленген орынға апаруға арналған шақырту. Әдетте позиция файл басынан басталған байттар санымен беріледі.
Get Attributes	Файл атрибутын алуға арналған шақырту, мысалы оның пайда болған уақыты.
Set Attributes	Файл атрибуттарын орнатуға арналған шақырту.
Rename	Файл атауын өзгертуге арналған шақырту. Бұл шақырту ОЖ әруақытта бола бермейді, өйткені файлды жаңа атаумен көшіріп, ескісін кезде де немесе деректер жинағына қатты сілтеме жасап, ескі қажет емес сілтемені өшіру арқылы осы нәтижені алуға болады.

Каталогтарға қолжетімділік үшін операциялық жүйе шамамен файлдармен жұмысқа арналған қызметтерді жүзеге асыратын жүйелік шақыртулар жинағын ұсынады (кесте-2.2). дегенмен каталог ішіндегі ақпаратты мәтін секілді оқу мүмкін болмағандықтан айтарлық өзгешеліктер бар.

Кесте-2.2. каталогтармен жұмысқа арналған жүйелік шақыртулар

Жүйелік шақырту	Сипаттама
Create	Тек элементтерден тұратын бос каталог жасалады «.» және «..»
Delete	«.» және «..» элементтерден басқа ешқандай басқа файлдар мен каталогтары жоқ бос каталогтар өшіріледі.
OpenDir	Бұл шақырту кез-келген каталогпен жұмыс басталмай тұрып орындалуы керек. Шақыртудың негізгі мақсаты — оперативті жадыға каталог параметрлерін жазу. Жалпы алғанда бұл шақырту файлға арналған Open шақыртуына ұқсас.

Жүйелік шақырту	Сипаттама
CloseDir	OpenDir шақыртуымен оқылған, каталог параметрлерінің соңы арналып белгіленген жадыны босатады
ReadDir	Каталог элементін оқуға арналады. Файлға арналған Read шақыртуымен салыстырғанда бұл шақыртудың ақпарат бірлігі файлдардың қасиеттерін анықтайтын деректер құрылымы
Rename	Файлға арналған Rename шақыртуымен бірдей
Link	Каталогтағы файлға қатты сілтеме жасау
Unlink	Каталогтағы файлға арналған қатты сілтемені өшіру. Егер файлға соңғы сілтеме өшірілсе, бұл файлдың деректер жинағына қолжетімділік жоғалады, ал деректер қоры алып жатқан дисктегі кеңістік бос болады.

Файлдармен жұмыс жасау үшін пайдаланушы қандайда бір белгіленген тіл қолданады. Тілдің иілгіштігіне байланысты пайдаланушыға жоғарыда көрсетілген жүйелік шақыртылуға сәйкес тілдің синтаксикалық құрылысы ұсынылады немесе жүйелік шақыртулар бөлігі тіл құрылысымен жасырынады.

UNIX-жүйелерінде файлдармен жұмыс жасаудың әдеттегі тәсілдерінің бірі осы мақсатта UNIX командалық түсіндіруші және оларға ұсынылатын команда тілін пайдалану. Командалық түсіндіруші мүмкіндіктеріне келесі тарауда шолу жасалады.

2.5.

UNIX ЖӘНЕ WINDOWS ФАЙЛДЫҚ ЖҮЙЕЛЕРІН ҰЙЫМДАСТЫРУ ҚАҒИДАЛАРЫ

2.5.1. UNIX файлдық жүйелерін ұйымдастыру қағидалары

Қарапайым s5fs файлдық жүйе негізінде UNIX ОЖ файлдық жүйесінің ұйымдастырылуын қарастырамыз.

Дискіде деректер жинағын сақтау үшін UNIX-жүйелерінде келесі тәсіл қолданылады: дискте сақталатын әр деректер жинағы блоктарға бөлінеді (бір блок көлемі диск секторының көлеміне тең). Деректердің бүтіндігін қамтамасыз ету үшін қызметтік блоктарда оның ішіне кіретін блоктарға сілтеме болады.

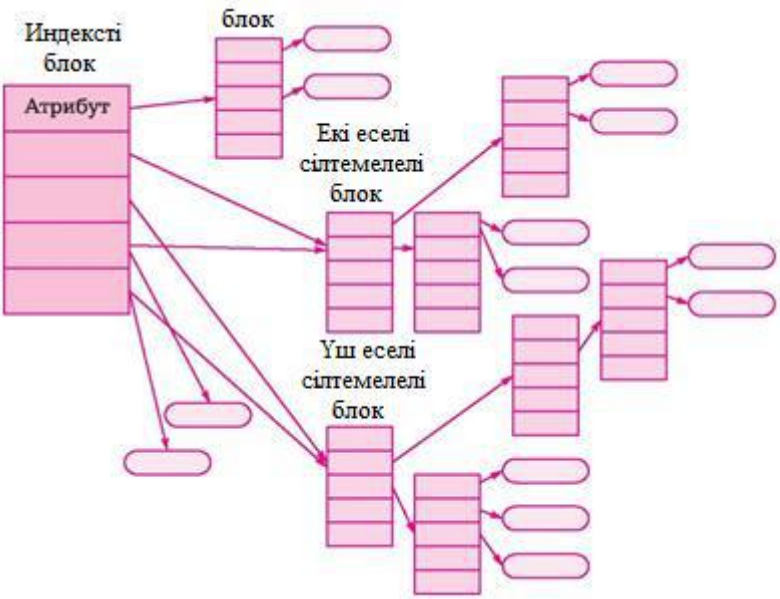
Бұл қызметтік блоктар бұтақ тәріздес құрылымға ұйымдастырылған, яғни әр блок басқа қызметтік блоктарға және пайдаланушы деректері блогына сілтеме жасалады (сурет-2.9).

Бұтақ тамыры индекстік блок (i-node), ол жерде файлдар атрибуттары (модификация уақыты, соңғы жүгіну уақыты, қолжетімділік құқығы, файл типі және т.б.) және деректер блоктарына сілтемелер массиві сақталады.

Сілтемелер массивінің көлемі шектелген. Егер блоктар саны массив көлемінен асып түссе жалаң сілтемелік блок жасалады да i-node элементтерінің біреуі соған сілтемені жасай бастайды. Жалаң сілтемелік блок деректер блогына сілтеме жасайды.

Сілтемелер массивінің де көлемі шектеулі болғандықтан индекстік блокта және жалаң сілтемелік блокта барлық сілтемелерді толтырып тастаған жағдайда қос сілтемелік блоктар жасалады. Дара блокқа жасалған сілтемелер екілікке ауысады да, индекстік блок қос блокқа сілтенеді.

Осылайша, сілтемелер ағашының тереңдігі өседі. Егер барлық деректер блогының сілтемелік блоктарына жинақтың ішіне кіретін *еселік* сілтемелер жетпесе,



Сурет-2.9 UNIX-файлдық жүйесінде деректер жинағының құрылымы

Үш есе сілтемелік блоктар жасалады.
өте терең деңгейге салынған сілтемелік блоктар болмайды, өйткені деректер блогын іздеуге жұмсалатын уақыт өте көп болып кетеді[16].

123	
4356	
12	File.txt
4356	Sudbir2

Сурет-2.10. Каталогты ұйымдастыру мысалы

Бұл файлдық жүйелердегі каталогтар арнайы файлды құрайды, оның ішінде файлдар мен каталогтар тізімі болады (2.10 сурет).
Бұл файлды әрқайсысы 16 байт көлемді жазбалар орналасқан, олардың ішіндегі ақпарат: бірінші екі байтта i-node файлының номері, ал қалған 14 байтта — файл атауы. Осылайша, бұл файлдық жүйелерде файл атауының ұзақтығы тек 14 символмен шектелген.

i-node номері астына тек 2 байт бөлінетіндіктен, каталогтағы файлдардың ең көп саны 65 535-санынан аса алмады. Бұл кестеде тек үш номер каталогтың өзін сипаттау үшін сонымен қатар оның аталық каталогын, алыстатылған файл (0 мәні) белгіленуін сипаттау мақсатында резервтелген, осы арқылы каталогтағы файлдың жалпы саны қысқарды.

Әр файлда өзінің жеке i-node болады. Ол жерде файл туралы барлық қызметтік ақпарат бар, оның көлемі, қолжетімділік құқығы және т.б. i-node жалпы құрылымы 2.3-кестеде келтірілген.

di_mode жолағы биттік жолақ, файлға қолжетімділік түрін және құқығын анықтайды (сурет-2.11).

Кесте-2.3. i-node құрылым элементтері

Жолақ	өлшемдер, байт	Сипаттама
di_mode	2	Файл түрі және қолжетімділік құқығы
di_nlinks	2	Файлға қатты сілтемелер саны
di_uid	2	Файл иесінің сәйкестендіргіші
di_gid	2	Топ иесінің сәйкестендіргіші
di_size	4	Файл өлшемі, байт
di_addr	39	Жады блоктарының адрестері бар индекстік блок
di_atime	4	Файлға соңғы қолжетімділік уақыты
di_mtime	4	Файлдың соңғы модификациялану уақыты
di_ctime	4	Индекстік дескриптордың соңғы өзгеру уақыты

Тип 4 бит	SUID	SGID	Sticky	User read	User Write	User exec	Group read	Group write	Group exec	Others read	Others write	Others Exec

Сурет-2.11. di_mode жолағы

Файл түрі алғашқы төрт битпен беріледі, және файл блоктық құрылғының тұрақты файлы ма, каталогы ма сол анықталады. Басқа биттер жүйе пайдаланушылары файлына қолжетімділік құқығын анықтайды, сонымен қатар арнайы орнатылған құқықтардың бар немесе жоқтығы туралы деректерді сақтайды.

Бұл файлдық жүйе қарапайымдылығымен ерекшеленеді. Дегенмен бұл қасиеті сенімділік, өнімділік және жан-жақты қызмет көрсету белгілі бір мәселелерінің теріс жағы болып табылады. Файлдық жүйеде деректердің жоғалуы немесе файлдық жүйе құрылымының бұзылуы орын алған жағдайда жады блогының қосымша адресаттары қолданылады. Тағы бір өзекті кемшілігі файл атауына және каталогтағы максимал файлдар санына қойылатын шектеу.

Linux тобының операциялық жүйелерінде соңғы уақытқа дейін негізгі файлдық жүйе ext2fs болды. Ол Linux операциялық жүйелерінің ең алғашқы нұсқаларында қолданылған Minix операциялық жүйесінің файлдық жүйесінің орынын алмастыруға келді. Ол 16-биттік архитектурасының шектеулілігі біліне бастаған, 1990 жылдарда ескіріп қалған s5fs файлдық жүйесіне қатты ұқсады. Сол себепті бірінші кеңейтілген extfs файлдық жүйе ойлап шығарылды, одан кейін ext2fs оның ізін жалғастырды.

Ext2fs файлдық жүйесі файл көлемінің шектеулілігін 2 Гбайт дейін кеңейтті және файл атауының ұзақтығын 255 таңбаға дейін арттырды. Одан бөлек, бұл файлдық жүйеде POSIX стандартының кейбір элементтері жүзеге асты. Өз негізіне байланысты ол классикалық s5fs файлдық жүйеге өте ұқсас.

Ол файлдық жүйенің ары қарай дамуы ext3fs болды. Оның алдындағы жүйелерден айырмашылығы журналдау механизмін қолдау болды.

Журналдау — кейбір қызметтік мәліметтерді арнайы файлға (журнал) жазу процесі. Ол ауытқу болған жағдайда файлдық жүйе құрылымын немесе оның ішіне болған мәліметтерді де қайта қалпына келтіреді. Заманауи файлдық жүйелердің көпшілігі журналдау және деректерді сақтаудың жоғары сенімділігін қамтамасыз етеді.

Журналданудың үш түрі бар: 1) writeback, тек құрылымдағы өзгерістер туралы ақпараттар сақталады, журналдағы жазба деректер жазбасымен синхрондалмаған;

2) ordered деп аталатын түрі writeback ұқсайды, бірақ журналға жазба файлда өзгерістер болғаннан кейін жүргізіледі; 3) journal, файлдық жүйе құрылымы туралы ақпаратпен бірге деректердің өзгерісі туралы да ақпараттар сақталады. Ең сенімді және ең баяу журналдау түрі journal болып табылады, ал сенімділік аз және тез журнал — writeback.

ext3fs файлдық жүйе журналданудың осы барлық түрлерін қолданды және ext2fs салыстырғанда әлдеқайда сенімді болды. Бірақ оның үлкен кемшілігі ext2fs қарағанда жылдамдығының өте төмен болуы.

Файлдық жүйе жұмысының жылдамдығымен мәселені шешу үшін және де басқа параметрлерін жақсарту үшін ext4fs файлдық жүйесі ойлап шығарылды. Бұл файлдық жүйе де журналданады, сонымен қатар, ол көлемі өте үлкен бөлімдерді қолдайды. Сонымен қатар оған экстенттерді сүйемелдеу қосылған, яғни файлдарға тек блоктанып бекітілген жады көлемін ғана емес, сипаттамасы үшін бір дискрептор қолданылатын, бірнеше блоктарды біріктіретін үлкен жады көлемін бөліп беру мүмкіндігі бар. Бұл файлдық жүйе ext3fs және оның алдында пайда болған файлдық жүйелерде шектеулі болған 32 000 каталогтарды жасауға мүмкіндік берді.

2.5.2. Windows файлдық жүйелерін ұйымдастыру процесі

Ең көне және ең танымал файлдық жүйелердің бірі FAT. Ең бірінші MS DOS операциялық жүйесі үшін жасалған файлдық жүйе қарапайымдылығына, жылдамдығына және тиімділігіне байланысты кең таралды.

Бұл файлдық жүйенің құрылымы өте қарапайым болды (сурет-2.12). алдымен жүктеу секторы орналасады, одан кейін — файлдарды орналастыруға арналған екі кесте (File Allocation Table — FAT, файлдық жүйенің атауын осылай етіп алынды, екінші FAT біріншісінің толық көшірмесі және оның зақымдалғанда қажет болады деген мақсатта сақталады), түбірлік каталог және файлдар.

Жүктеу секторы	ФЖ жайлы мәлімет	Қосымша секторлар	FAT#1	FAT#2	Түпкі каталог (FAT 12/16)	Мәліметтер

Сурет-2.12. FAT файлдық жүйесінің құрылымы

Бұл файлдық жүйеде барлық дисктік кеңістік кластерлерге бөлінеді, FAT кестесі әр кластерге қызметтік ұяшықтар береді. Файл туралы ақпарат тізімді білдіреді. Егер ұяшық файлдағы соңғы кластерге сәйкес болса ол FFFF мәнінен тұрады. Қолданылмаған кластерлер 0000 болып белгіленген, ал алыстағы кластерлер — FFF7. Файлды өшірген кезде тек каталогта жазба жасалады, ал кластерлер тізімі бұзылмайды. Егер кестеде ұяшықтардың тұтастық тізбегі бұзылмаған болса, файлды тез қалпына келтіруге мүмкіндік береді.

Бұл файлдық жүйенің беріктілігі жоғары. Файлдар және бос кластерлер туралы ақпараттар бір кестеде сақталады, сол себепті әдетте екі операция арқылы орындалатын файл жазбасы FAT ішінде бір әрекетпен орындалады.

FAT файлдық жүйесі бірнеше операциялық жүйелерді және бірнеше инкарнациядан өтті. Қазіргі таңда FAT төрт нұсқасы бар: FAT12, FAT16, FAT32 және exFAT. Осы уақытқа дейін файлдық жүйенің осы форматы, әсіресе тасымалдаушы флэш-жинақтауыш секілді деректерді жинақтауыштары үшін белсенді түрде қолданылады.

NTFS файлдық жүйесін Microsoft фирмасы 1990 жылдардың басында шығарды. Бұл жүйе көп жағдайда өз уақытында OS/2 операциялық жүйеде новатор болып қолданылған HPFS файлдық жүйесінің ұрпағы болып саналды. Бұл операциялық жүйе Microsoft және IBM фирмалары бірге ойлап шығарды. Бірақ одан кейін Microsoft фирмасы өзінің жекеменшік операциялық жүйесін Windows дамыту туралы шешім қабылдады және HPFS файлдық жүйесінің зерттемесі негізінде оған арналып жаңа файлдық жүйе жобаланды.

Windows 3.xx және Windows 9x ядроларына негізделген Windows ескі нұсқалары, тек FAT файлдық жүйесімен жұмыс жасады. FAT өте көп модификациясы бар болғанымен және олардың жұмыстарының жоғары жылдамдығы шағын файлдардың көп санымен жұмыс жасауымен ерекшеленсе де олардың басым бөлігі UNIX/Linux-жүйелеріне жол береді.

FAT ұрпағы файлдық жүйелерімен байланысты негізгі мәселелер келесідей:

- Файлдық жүйелер FAT файлдық жүйенің жалпы өнімділігін бәсеңдететін деректер фрагменттеуге өте сезімтал;
- Логикалық бөлімдердің айтарлықтай үлкен көлемдеріне дисктік кеңістікті шығындайды.
- кенет ақау болған кезде немесе қайта жүктелу кезінде файлдық жүйенің жұмыс қабілеттілігін қайта қалпына келтіруге мүмкіндік беретін қателерден қайта қалпына келтіру механизмдерін қолдамайды;

- бірнеше ұйымдар мен серіктестіктерге Windows операциялық жүйесін пайдалануға кедергі жасаған POSIX қолдауы жоқ;
- Пайдаланушылар арасында деректерді бөлу құралы және қолжетімділік құқығын басқару құралдары жоқ.

NTFS файлдық жүйе FAT файлдық жүйесіне тән кемшіліктерді ескере отырып жасалды. Осы файлдық жүйеге тән негізгі қасиеттер:

- Сенімділік және қайта қалпына келушілік. NTFS файлдық жүйесі кіріс/шығыс операциялары транзакцияны білдіретіндей жобаланған. Бұл транзакциялар бөлуге келмейді, яғни талап етілген операция толық аяқталады немесе мүлдем аяқтамайды. Бұл файлдық жүйе күйін ақау болған жағдайда кейін қайтару операцияларын ауырсындырмай жасауға мүмкіндік береді;
- Қауіпсіздік және деректерге қолжетімділікті бақылау. NTFS файлдық жүйесі Windows қауіпсіздігінің жалпы архитектурасына сәйкес файлдар мен каталогтарды қорғалған нысан ретінде түсіндіреді. Бұл мәліметтерге белгілі бір пайдаланушылардың немесе пайдаланушылар тобының қолжетімділігін шектеуге мүмкіндік береді;
- Дисктік кеңістікті тиімді бөлу. NTFS файлдық жүйесі дисктік кеңістікті басқарудың механизмдердің тұтас қатарын қолданады, олардың ішінде каталогтар мен дисктерді қысуды қолдау, кесілген файлдармен жұмысты қолдау, мәліметтерді дефрагменттеу үшін мамандандырылған API бар болуы;
- POSIX қолдауы, қатты сілтемелер, ұзын атаулар, мәліметтерді шифрлеу, көлемі Гбайт көп логикалық томдармен жұмыс жасауды қамту.

NTFS файлдық жүйесі құрамында бірнеше жүйелік аумақ және деректерді сақтау алаңы бар. NTFS файлдық жүйесі жалпы құрылымы 2.13-суретте көрсетілген.

Негізгі жүйелік алаңдардың бірі Master File Table (MFT) алаңы болып табылады. Бұл алаңда жүйедегі барлық файлдар туралы ақпарат бар, MFT өзі туралы ақпаратта кіреді. Қандайда бір жаңа файл жасалғанда немесе жүйеден өшірілген кезде, сәйкес өзгерістер MFT ішінде де жүреді.

MFT кестесіндегі алғашқы 16 жазба қызметтік файлдар туралы ақпараттарды сақтау үшін резервтелген. Кестедегі алғашқы жазба MFT өзі туралы ақпаратты сақтауға арналған.

Жүктемелік	Мастер File Table	Резерв	Жүйелік	Қарапайым	MFT алғашқы 16	Қарапайым
Сектор		MFT	Файлдар	файлдар		файлдар

Сурет-2.13 NTFS файлдық жүйесінің құрылымы

Екінші жазба кестесінің көшірмесі туралы ақпараттан тұрады. Бұл көшірме әдетте диск ортасында орналасады да негізгі MFT зақымдалған болса, қайта қалпына келтіру үшін қолданылады. MFT кестесіндегі үшінші жазба файлдарды қайта қалпына келтіру үшін қолданылатын жүйелік журнал. 4 бастап 16 дейінгі жазбалар басқа қызметтік файлдар туралы ақпараттардан тұрады, ал 17 және одан кейінгі жазбалар қарапайым файлдар туралы ақпараттарды сақтауда пайдаланылады.

Қызметтік файлдарға мета деректер файлдары деп аталатын файлдар жатады. Метадеректер файлдары файлдық жүйені өшірген кезде автоматты түрде пайда болады. Мета деректердің негізгі файлдары және олардың сипаттамалары 2.4-кестеде келтірілген.

NTFS файлдық жүйесінің қызықты ерекшеліктерінің бірі файлдардағы ақпарат ағынын қолдау. *Осы ретте,* файлмен ақпарат ағынының біреуі емес бірнешеуі байланыстырылуы мүмкін. Бұл файлда тек негізгі деректерді сақтап қана емес ұқсас деректер ағынына қосымша ақпараттарды сақтауға рұқсат береді.

Кесте-2.4. NTFS мета деректерінің файлдары

Файл Метадеректер	Сипаттама
$MFT	Master File Table таныстырады
$MFTmirr	Диск ортасында орналасқан алғашқы 16 жазбаның MFT көшірмесі
$LogFile	Файлдық жүйелерді қайта қалпына келтіруге арналған файлдық операциялардың жүйелік журналы
$Volume	Деректер томы туралы ақпарат — белгі, файлдық жүйенің нұсқасы және т.б.
$AttrDef	Файлдардың стандартты атрибуттарының тізімі
$.	Түбірлік каталог
$Bitmap	Томдағы бос орын картасының орны
$Boot	Жүктемелі сектор
$Quota	Пайдаланушыларға арналған дисктік кеңістік шектеуліктері туралы ақпарат ақпарат сақталатын файл
$Upcase	Файлдар атауындағы үлкен және жазба әріптерінің сәйкестік кестесі. Файлдар атауындағы Unicode қолдауымен байланысты

Мысалы, файл автор туралы ақпарат, көшіру құқығы және т.б. файлға қатысты әр ағын тәуелсіз ашыла алады. Бір файлмен байланысқан барлық ағындар, бір атрибуттарға ие.
Ағындарды пайдалану мысалы ретінде келесі листингті қарастыруға болады:

```
C:\>echo data > data.txt C:\>echo
data1 > data.txt:stream1 C:\>echo
data2 > data.txt:stream2 C:\>more <
data.txt data
C:\>more < data.txt:stream1
data1
C:\>more < data.txt:stream2
data2
```

data.txt файлының негізгі ағыны ішінде data хабарламаның бар екені көрініп тұр, екі баламалы деректер ағыны өздерінің меншікті ақпараттарынан тұрады. Бірақ файлдық жүйеде бұл ақпараттар ағынының барлығы бір-дара файл болады.

Файлдарды, каталогтарды және бөлек томдарды автоматты түрде қысу NTFS файлдық жүйесінің ерекшелігі болып табылады. Қысылған файлдар Windows кез-келген қосымшасымен қысылмаған кезіндегідей өзгертіледі/оқылады, яғни файлдардың қысылуы және ашылуы нақты уақыт масштабында жүзеге асады. Дегенмен қысылған файлдармен, каталогтармен және томдармен жұмыс жасау қысылмаған файлдарға қарағанда көп уақытты қажет етеді.

Қолдауы файлдық жүйе NTFS енгізілген тағы бір механизм туралы айтып кеткен жөн — қатты сілтемелермен жұмыс. Бұл механизм POSIX стандартын қолдайтын жалпы механизмнің бөлігі ретінде енгізілген және бір дерекке сілтемеленетін әр түрлі файлдар жасауға мүмкіндік береді. Оған қоса ол мәліметтер қайталанбайды, дисктік кеңістікті үнемдейсіз. Қатты сілтемеден біреуін өшірген жағдайда деректердің өздері өшірілмейді. Деректер толық өшіріледі егер олармен байланысты қатты сілтеме өшірілген болса.

Windows қатты сілтемелер жасау үшін fsutil командаңың hardlink create параметрімен қолдануға болады. Бұл команда қолданыста бар файл үшін қатты сілтеме жасай алады.

Қатты сілтеме жасау мақсатында бұл команданы шақыртудың жалпы форматы келесідей:

fsutil hardlink create қатты_сілтеме_атауы бар_файлдың_атауы.

Жалпы алғанда NTFS өте бір қуатты және сенімді файлдық жүйе болып саналады. Бойында дисктік жинақтауыштарда мәліметтерді

сақтауда және өңдеуде сенімділікті нығайтып, жұмысын жеңілдететін қасиеттері бар. Және Microsoft корпорациясының келесі операциялық жүйелері бұл файлдық жүйені қолданудан бас тартқалы отырған жоқ. Болашақ Windows-жүйелеріне пайдалануға арналған. Өңделген файлдық жүйе WinFS, тәуелсіз болмайды. Ол NTFS файлдық жүйесіне қондырма бөлім ретінде шығатын деректердің реляциялық базасын білдіреді.

Б
ҚЫЛАУ СҰРАҚТАРЫ

1. файлдық жүйе қандай негізгі қызметтер атқарады?
2. Жинақтауышта деректердің логикалық және физикалық құрылымының негізгі ерекшеліктері неде?
3. NTFS деректерінің құрылымы қандай?
4. *Ата-ана каталог*ы дегеніміз не?
5. Ағымдағы каталог дегеніміз не?
6. Файлдардың абсолюттік және қатыстық атауларының айырмашылықтары қандай?
7. Операциялық жүйелердің қандай атқарымдары қолданбалы бағдарлама мен деректерді жинақтауышта физикалық ұйымдастыру арасында байланысты қамтамасыз етеді?
8. Файлдың қатыстық атауы абсолюттік атауынан ұзақ болуы мүмкін бе?

ТАРАУ 3

ОПЕРАЦИЯЛЫҚ ЖҮЙЕЛЕРДІҢ ЖАДЫН БАСҚАРУ

3.1. ЖАЛПЫ ТҮСІНІК

Көбінесе оперативті жады деп аталатын жады кез-келген есептеуіш жүйесінің маңызды ресурстарының бірі. Сол себепті жадыны басқаратын қосалқы жүйе операциялық жүйенің маңызды қызметі. Бағдарламалар есептеуіш жүйенің процессорымен орындалуы үшін бағдарламаның орындалатын коды негізгі жадыға жүктелген болуы тиіс.

Негізгі жадыда деректердің өңдеудің жылдамдығы әдетте бағдарламаларды сақтайтын сыртқы жинақтауыштарда деректерді өңдеу жылдамдығынан айтарлықтай тез. Сол себептен операциялық жүйені жасау барысында жүктеме процесін және бағдарламаның орындалуын әр орындалып отырған процеске оперативті жадыдан жеткілікті орын жететіндей оңтайландыруға тырысады. Бұл біріншіден ағымдағы процестің орындалуын жылдамдатады, екіншіден операциялық жүйенің басқа процестерінің жұмысын баяулатпайды.

Операциялық жүйе оперативті жадының шектелген көлемін бірнеше мақсатта (процестерге) бөлуі қажет болған кезде мәселелер туындайды. Сонымен қатар, әрқайсысы ағымдағы есептеулерді орындауға жеткілікті бөлігін ғана алады, бірақ ең аз жады бағдарлама жұмысына қажет. Тек тапсырманың белсенді емес бөлігі ғана сыртқы жадына орналасады да, ол жерден қажет болған жағдайда алынады.

Осыдан операциялық жүйенің жадын басқару жүйесін жасағанда қажет негізгі мақсаттар туындайды:

1) Негізгі жадыға қолжетімділік уақытын қысқарту;
2) Процестерге қолжетімді негізгі жадының көлемін арттыру;
3) Негізгі жадыға процестер қолжетімділігінің тиімділігін арттыру.

Жадыны басқару жүйесінің шешетін мәселелері келесідей:

1) Процестерге негізгі жадыны бөлу;
2) Процестің адрестік кеңістігін жадының белгіленген негізгі алаңында көрсетеді;

3) Негізгі жадының қолжетімді көлемін пайдалана отырып деректерге қолжетімділік уақытын азайту.

Жадының динамикалық бөлінуі шартында осы мәселелерді шешу үшін жады менеджері процес үшін бөлінетін барлық жады блоктарының тізімін қолдап отыруы қажет. Бұл тізім деректердің тізіммен байланысқан кез-келген құрылымы көмегімен сүйемелдене алады. Ең қарапайым жағдайда бұндай құрылым жады блоктарының дескрипторларын көрсетуі мүмкін, әр дескриптор келесі сәйкес жады блогына және оның көлеміне сілтеме жасайды. Жады менеджері бұл құрылымның тұтастығын қамтамасыз ету мақсатында жадыны басқару ережелеріне сай белгілі бір ретпен тізімнен элементтерді алып тастайды немесе керісінше тізімге кіргізеді. Процестер арасында жады көлеміне көбірек ие болу мақсатымен өзара бәсекелестікке түсетін жадыны бөлудің бірнеше стратегиялары бар:

1) «ең қолайлы аймақ» стратегиясы негізінде бөлу; «ең қолайлы аймақ»;

2) «қолайлылығы төмен аймақ» стратегиясы негізінде бөлу;

3) «бірінші қолайлы аймақ» стратегиясы негізінде бөлу;

4) «келесі қолайлы аймақ» стратегиясы негізінде бөлу.

Бірінші стратегияны пайдалану кезінде менеждер процеске бос жадының көлемі талап етілген көлемнен асып түсетін қол жетімділігі ең аз блогын бөледі. Бұл стратегия жадының барлық бос блоктар тізімінен көлемі бойынша сәйкес келетін блок іздеуге көп уақытты талап етеді. Оның шешімі жады блоктарының тізімін көлем бойынша саралау. Ең үлкен ұқсастық қағидасы бойынша жадыны ерекшелеу тәсілімен байланысты өте үлкен мәселе бар. Осы стратегияны пайдаланған кезде жадының пайдалануға көлемі аз көптеген бөліктері қалып қояды да олар мүлдем пайдаланылмай қалады. Бұл жағдай жадты фрагменттеу деп аталады және осы ресурсты текке шығындауға әкеледі. Жадының фрагменттелуіне қатысты мәселе жадыны басқару тәсілдерінің бәрінде бар, бірақ «ең қолайлы аймақ» стратегиясында көп деңгейде орын алады. Фрагменттеуді болдырмау үшін сұралатын жады көлемінен әлдеқайда көп көлем бөлуге болады. Мысалы, әр сұралатын блок көлемін белгілі бір өлшемге теңестіруге (әдетте екі дәрежесіне тең) және үлкен көлем бөлуге болады. Дегенмен бұл жерде жадының фрагменттеу мәселесі шешілмейді, бұл тәсіл тек фрагменттеуді жасыруға мүмкіндік береді.

Кемшіліктерге қарамастан, жадты фрагменттеуді азайтуға «қолайлылығы төмен аймақ» стратегиясы көмектеседі. Бұл тәсілдің мәні әрқашан жадының ең үлкен бос алаңы ізделеді де, дәл сол алаң қолдануға беріледі. Бұл стратегияның негізгі түйіні мынада: бірінші жағдайда фрагменттеуге алып келетін пайда болған жадының

қолданылмайтын алаңының көлемі бірінші стратегияны пайдаланғанға қарағанда өте үлкен. Егер «ең қолайлы аймақ» стратегиясы негізінде жадының бос, пайдаланбай қалып қойған бөліктері («тесік») қайта пайдалану үшін көлемдері өте аз болып, жадты фрагменттеуді туындататын болса, екінші тәсілді пайдаланғанда пайда болатын «тесіктер» жадты бөлуге сұралатын келесі сұратылымдарда пайдаға асады, фрагменттелуі мүмкіндігін азайтады.

«Бірінші қолайлы аймақ» стратегиясын қолданған кезде жады менеджері көлемі бойынша сәйкес келетін бірінші блокты іздеу мақсатында бос блоктар тізімін сканерлейді. Өз қарапайымдылығына қарамастан бұндай бос блоктарды іздеу алгоритмі «ең қолайлы аймақ» салыстырмалы түрде өзін жақсы көрсетеді, оны пайдаланған кезде жады фрагменттелуге аз ұшырайды. Бұл стратегияның басты кемшілігі жадының «тесіктерінің» көп бөлігі бос блоктар тізімінің басына шоғырланады, сол себепті жады менеджері уақыт өткен сайын бос блоктарды тізімнің бас жағынан емес орта жағынан іздеуге мәжбүр болады.

«Бірінші қолайлы аймақ» стратегиясының негізгі мәселесін шешу үшін «келесі қолайлы аймақ» стратегиясы қолданылады. Жадының шағын бөліктерін тізім басына шоғырлануын болдырмау үшін бос блоктардың сақина тәріздес тізімі қолданылады. Сонымен қатар, жады блоктарын таңдау үшін «бірінші қолайлы аймақ» стратегиясы қолданылады, бірақ блокты таңдаған әр таңдаған сайын тізім басы таңдалған блоктан кейінгі блокты көрсете бастайды. Оған қоса, фрагменттер бос блоктардың тізімі бойынша азды көпті тегіс орналасады.

Бос блоктарды таңдау алгоритмі қаншалықты айлақор болса да жадының толық фрагменттеуді қамтамасыз ететін абсолютті стратегияны ойлап табу мүмкін емес. Ерте ме кеш пе жады да «тесіктер» саны сыни мәннен асуын бастайды, және бұл жағдай қосымша өңдеуді қажет етеді. Бұндай өңдеу қалыңдату немесе жадты дефрегменттеу деп аталады.

Дефрагменттеу процесі «тесіктер» мәселесін шешеді. Ол барлық қолданыстағы жады блоктарын жадының бір бөлігіне жинайды да, «тесіктерді» бөлек үлкен бос блокқа құяды. Бұл операцияның шығыны көп, өйткені ол барлық процестердің тоқтап тұруын талап етеді, адрестердегі қажет жылжытуларды есептеу, деректерді физикалық тасымалдау және т.б. Осыған қарамастан бұл тәсіл процестерді шақыру уақытына сыни талап қоймайтын кейбір операциялық жүйелерде қолданылады, есесіне бағдарламалау жүйесінің және бағдарламалардың өздерінің жұмыстарын жеңілдетеді.

3.2.
ВИРТУАЛ ЖӘНЕ ФИЗИКАЛЫҚ ЖАДЫ

Физикалық жады кез-келген есептеу жүйесінің аппараттық құралдарының құраушы бөліктерінің бірі болып табылады. Ол операциялық жүйені жүктеу үшін және пайдаланушылардың, жүйелердің процестерін орындау үшін қолданылады. Қазіргі таңда бұл жады бірінші кезекте RAM бірнеше чиптерінен тұратын жадының бірнеше модульдерін білдіреді. Бұл жады өте тез саналады (ол жылдамдығы жағынан тек регистрлік және процессордың кэш-жинақтауышынан артта қалады), дегенмен өте қымбат және жадының бұл түрінің көлемі әдетте өте шектелген.

Осы шектеулердің әсерінен егер бір мезетте процестердің көп бөлігі жұмыс жасап тұрса физикалық жады тез таусылады. Осы мәселені шешу үшін виртуал жады қолданылады. Виртуал жады бірнеше жадыларды (RAM рұқсаты бар жады, өте баяу дисктік жады және т.б.) біріктіруге, үлкен бағдарламаларды іске қосуға мүмкіндік береді. Бұл бағдарламаларға арналған барлық жады виртуалды жадының бірыңғай массиві түрінде ұсынылады.Ол қарапайым физикалық жады секілді, айырмашылығы оның көлемі үлкен. Осылайша, виртуал жады жадының логикалық құрылымын оның аппараттық ұйымдастыруынан айыруға мүмкіндік жасайды.

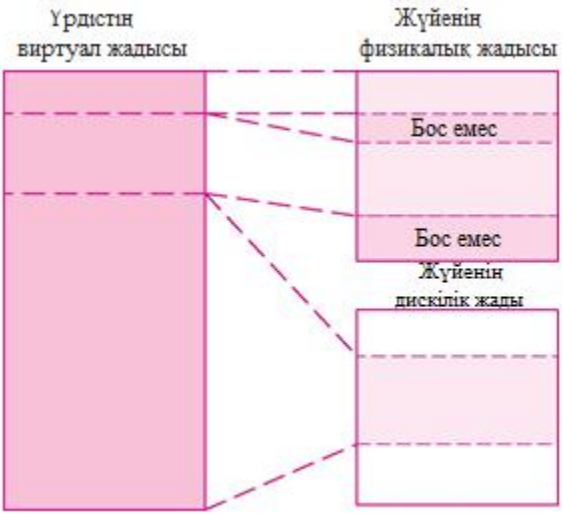
Виртуал жадыны пайдалану виртуалды адресацияны пайдалану есебінен физикалық жадтың фрагменттелу мәселесін шешуге көмектеседі.

Осы механизмді пайдалану кезінде әр процеске виртуалды жадыдан тәуелсіз аумақ бөлінеді, ол процестерді бір бірінің ықпалынан қорғайды (сурет-3.1). сонымен қатар виртуал жады мен логикалық адресті қолдану жадыдағы процестерді базалық адрестерді өзгерту есебінен рұқсат береді.

Виртуал жады негізгі жадыға олардың адрестік кеңістігінің тек бір бөлігі ғана бейнеленетін жағдайында процестердің орындалуға мүмкіндігі бар. Барлық өмір айналымы талап етілетін жадының аумағы негізгі жадыға жүктеледі, ал көп уақыт қажет болмаған аймақтар анағұрлым баяу дисктік немесе басқа қосымша жадыға жіберіледі.

Бұл механизм негізгі жадының физикалық көлеміне қарағанда үлкен көлемді бағдарламаларды орындауға жағдай жасайды. Сонымен қатар ол операциялық жүйенің көп процессорлық жұмыс режимін қолдау үшін физикалық жадыда бір мезгілде бірнеше процестерді (немесе, кем дегенде олардың маңызды бөлімдерін) орналастыруға рұқсат береді. Виртуалды жадыны негізгі жадыдан қосымша жадыға ауыстыру процестерін басқарудың негізгі элементтері ретінде тарту механизмдері қолданылады, ал орын ауыстыратын блоктардың көлемін анықтау үшін негізгі жадыға, негізгі жадыдан процестерді ауыстыру немесе виртуал жадының сегменттік немесе парақтық тарту қолданылады.

Процестердің орнын ауыстырған кезде бұғатталған процестер негізгі жадыдан толықтай қосымша жадыға ауыстырылуы мүмкін. Сонымен қатар, процестің барлық деректері және оның коды толықтай ауысады, негізгі жадыны босатады. Ал процестің өзі белсенді ету қажеттілігі пайда болғанша белсенді емес күйге түседі. Белсендіру қажеттілігі пайда болғанда процестің коды және оның барлық деректері қосымша жадыдан кері қарай негізгі жадыға қайта көшіріледі, басқа бір процестің негізгі жадыдан қосымша жадыға алмасуы нәтижесінен болуы да мүмкін. Осыдан кейін үрдіс белсенді күйіне оралады да қай жерінен тоқтаған болса сол жерінен ары қарай нұсқаулық бойынша орындалуын жалғастырады.



Сурет-3.1. Үрдістің виртуалды жадысының физикалық жадыға

Бұл тәсіл негізгі жадыда бір мезетте бірнеше процестердің орындауына жағдай жасайды, бұл процестер үшін қосымша жадыда орналасқан деректердің жүктелуіне қатысты орын алатын кешеуілдеулер болмайды. Белсенді процестердің әсерінің жылдамдықтарын арттыруға мүмкіндік беріп, барлық деректері оперативті жадыда орналасады. Дегенмен процестерді белсенді күйден белсенді емес күйге ауыстыру операциясы айтарлықтай ұзақ болып табылады, өйткені процестердің барлық деректерін толықтай жүктеп, қайта жүктемені түсіру керек. Сонымен қатар, бұндай тәсіл процес және оның деректері көлеміне қосымша шектеулер қояды: бір процес коды және деректер көлемінің қосындысы оперативті жадының қолжетімді көлемінен асып кетпеуі керек.

Виртуалды жадыны қолданудың бұндай тәсілі виртуалды жады механизмдерінің ең бастапқы кезде жүзеге асуы кезінде қолайлы. Одан кейін виртуалды жадымен сегменттер негізінде басқару алгоритмдері кең таралды, ал қазіргі уақытта жады парақтары негізінде басқару тәсілі қолданылады.

3.3.

ЖАДЫНЫ СЕГМЕНТТІК ЖӘНЕ ПАРАҚТЫҚ ҰЙЫМДАСТЫРУ

Виртуал жадыны сегменттік ұйымдастыру келесі тәсілге негізделген: бағдарламаны орындайтын жадының адрестік кеңістігі бір-бірімен қиылыспайтын аймақтарға бөлінеді, ол сегмент деп аталады. Сегменттердің көлемдері әр түрлі болуы мүмкін. Виртуалды жадыны ұйымдастыру кезінде адресация екі мән көмегімен жүзеге асады, біріншісі сегмент номері, екіншісі — сегмент ішінде ығысу.

Бұндай сегменттерді бағдарлама жасаушылар өздері таңдай алады (яғни қолмен), сондай-ақ орындалып жатқан бағдарламаны құрастыру кезінде автоматты түрде линкормен таңдала алады. Мысалы, осындай сегменттер ретінде код, дерек, константа және т.б. сегменттері қолданыла алады. Бұндай сегменттер егер олар қазіргі уақытта қажет болмаса негізгі жадыдан қосымша жадыға жүктеле алады, егер процес жұмысы үшін қажет болса қосымша жадыдан негізгі жадқа қайта

жүйеленеді.

Бағдарламалаушы тұрғысынан виртуалды жадының осындай ұйымдасуы өте ыңғайлы болып табылады, өйткені ол өзі бағдарламаларға модульдік құрылым ұсынады. Әр сегментпен көлемі, сегмент атрибуттары туралы ақпараттан тұратын оның дескрипторы *ассоциацияланған*. Егер жады аумағы орналасқан сегмент атрибуттарымен рұқсат етілген болса және ығысу сегмент көлемінен аспаса процес жадының қандай да бір бөлігіне жүгіне алады. Ондай жағдай орын алмаса сегмент шекарасының бұзылғаны туралы ақпарат түрлендіріледі.

Оның үстіне дескриптор жадының қай жерінде сегмент орналасқаны туралы, негізгі жадыда сегменттің бар немесе жоқтығы туралы ескертетін қатысу жалауы туралы ақпараттан тұрады. Егер негізгі жадыға жүктелмеген сегментке жүгіну талпынысы орын алса, ерекшелік генерацияланады. Операциялық жүйе қажет сегментті негізгі жадыға жүктейді, сегмент дискрепторындағы параметрлерді күйге келтіреді және сегментте жадыға жүгіну операциясын орындайды.

Виртуалды жадыны ұйымдастырудың осы қарастырылған түр фрагменттелуді азайтады. Сонымен қатар ол қолданылмаған сегменттерді жүктемеден шығару есебінен негізгі жадыны үнемдеуге көмектеседі, сегменттер көмегімен бөлінетін бағдарламалар модульдерінің мәселелерін шешеді. Мұндай модульдер сегмент жадысына жүктеледі, ал процестерде пайдаланылып жатқан сегменттер дискрепторларының кестесіне осы жалғыз сегментке сілтеме жасалады.

Виртуалды жадыны осылай ұйымдастырудың кемшілігі бұл модельдің адресация операциясы өте баяу, өйткені бірінші сегмент дескрипторына рұқсат алу керек, жадыдағы сегменттің орналасқан жерін оқу, одан кейін ығысуды пайдаланып соңғы адресті анықтау қажет. Сонымен қатар жады және процессорлық уақыт сегменттер дескрипторларына және дескриптор кестесіне, олардың орналасуы және сегменттерді жүйелеу кезінде адрестердің жаңаруына жұмсалады.

Осы мәселелерді шешу үшін заманауи операциялық жүйелерде виртуалды жадыны ұйымдастырудың парақтық тәсілі қолданылады. Парақтық ұйымдастыру кезінде виртуалды жадының барлық кеңістігі жады парақтар деп аталатын белгіленген көлемі бар блоктарға бөлінеді. Белгілі бір көлемде жады қажет болған кезде бұл өлшем парақтың өлшемімен теңестіріледі де жады парағының толық саны бөлінеді. Егер парақтың қандай да бір бөлігі пайдаланылмаған болса, онда жады ары қара таратылмайды да ары қарай пайдалануға мүмкіндік болмайды. Бұл жағдайда ішкі фрагменттеу сөз болады.

Виртуалды жадыны осылай ұйымдастырған кезде негізгі жады да «жады парағының фреймдері» деп аталатын виртуал жадының парақтарының көлемдерімен бірдей бөлінеді. Бұндай фреймдер виртуал жадының парақтары жатқызылуы қажет жиектеме секілді. Осындай парақтар бір-бірінен тәуелсіз қосымша жадқа қажеттігінше көшіріле алады және қажет болған жағдайда негізгі жадының қолжетімді фреймге қайта жүктеледі.

Виртуалды жадыны осылай ұйымдастырудың артықшылығы дәл сегменттік тәсіл секілді бұл тәсіл де қосымша жадыны пайдалану арқылы негізгі жадыдағы қолжетімді көлемді арттыра алады. Бір процес үшін әр уақыт сайын оған тиесілі емес жады парақтарын емес тек өзіне қажеттісін ғана жүктеу қажет. Басқалары қосымша жадыға көшіріле алады, басқа деректер мен процестерге негізгі жадыдан орын босатады.

Егер дәл сол сәтте негізгі жадыдан қосымша жадыға көшірілген жады парағына өтінім жіберілсе, парақша қателігі генерацияланады. Осындай қатені алған менеджер сұратылған парақты негізгі жадының бос фрейміне жүктейді. Қажеттігінше мұндай фрейм басқа пайдаланылмай тұрған парақтарды шығарып тастау жолымен босатылады. Осындай операциялардың жиілігі процестер пайдаланып жатқан парақ көлеміне және парақтардың жалпы санына тәуелді. Іс жүзінде негізгі жадыда болуы қажет минималды парақтар саны бар. Бұл санды «парақтардың минималды жұмыс жиынтығы» деген атауға ие және процес пайдаланып отырған жадының максимал өлшеміне тәуелді анықталады.

Жадының қай парақшасы негізгі жадыдан қосымша жадыға ауысуын анықтау үшін «парақтарды ауыстыру стратегиясы» деп аталатын арнайы алгоритм қолданылады.

Мұндай стратегиялардың жүзеге асуы үшін келесі алгоритмдер қолданыла алады.

Алгоритм LRU (Least recently used — бәрінен көп пайдаланылмайтын парақ). Бұл алгоритмді қолданған кезде негізгіден қосымша жадыға барлық парақтар ішінен ең ұзақ пайдаланылмаған парақ ауысады. Осы алгоритмді жүзеге асыру үшін әр параққа жүгінген кезде уақыт белгісі қойылады. Бұл жағдайда операциялық жүйе уақыт белгісі ең ұзақ пайдаланылмаған парақты табу үшін барлық парақтарды сканерлеп шығуы қажет.

Бұл мәселе шешімінің баламалы нұсқасы уақыт мәндерінің белгіленуі бойынша сұрыпталған парақтар тізімі болуы мүмкін.

Алгоритм NRU (Not recently used — біраз уақыт пайдаланылмаған парақ). Бұл алгоритм LRU алгоритміне өте ұқсас, бірақ аз шығынды қажет етеді. Жадының әр парағы арнайы битпен қамтылған, ол параққа

жүгіну болған жағдайда 1 мәніне ие болады. Қандай да белгілі уақыт сайын операциялық жүйе барлық парақтарды осы битті 0 лақтырып отырады. 0 мәніне ие биті бар ір парақ негізгі жадыда орын босату үшін қосымша жадыға жүктеліп шығарылып тасталуы мүмкін.

Алгоритм FIFO (First in, first out — парақтар кезегі). Осы алгоритмді пайдаланған кезде ұзақ уақыт бойы негізгі жадыда болған және жақын уақыт ішінде қажет болмайтын парақ қосымша жадыға ауысады. Осы стратегияны орындау үшін жадының барлық парақтары негізгі жадыда болған уақыты бойынша тізімге келтірілген. Тізім басында орналасқан парақ негізгі жадына бірінші болып ауысады, ал негізгі жадыға қосымша жадыдан ауысып келген парақ тізімнің ең соңына орналастырылады. Дегенмен бұл тәсіл жады жүйесінің өнімділігін төмендетуі ықтимал, өйткені оны пайдаланған кезде парақтарға жүгіну жиілігі ескерілмейді. Бұл жадтың жиі қолданылатын парақтары жалпы ережеге сәйкес қосымша жадыға ауысып, парақша қателіктерін болдырмау генерациясының көп санын шақыруы мүмкін. Ал ол өз кезегінде қосымша жадыдан жиі пайдаланылатын қажетті парақты *тартуды* шығаруды туындатады.

***Айналма* алгоритм.** Тәсіл FIFO стратегиясының бір нұсқасы, өзгешелігі барлық парақтар сақиналық тізімге біріктірілген. Параққа сұратылым келген кезде 1 мәні орнатылатын сілтемелі битке *ұқсастырылған*. Негізгі жадыдан қосымшаға парақ ауыстыру қажет болған сайын менеджер сілтемелі бит 0 мәніне ие парақ кездеспейінше тізімді аралап қарайды. Оның үстіне әр қаралған парақ үшін бұл мән 0 түсіп отырады. Сілтемелік бит 0 тең парақша кездескен бойда ол парақша лезде қосымша жадыға көшіріледі.

Парақшаларды шығарып тастауды басқарудың басқа да алгоритмдері бар. Мысалы жады парақшасына өтінім келу жиілігін ескеретін жиілік алгоритмі бар. Сонымен қатар шығарып тастау үшін парақшаны кездейсоқ таңдау тәсілі де қолданылуы ықтимал.

Ескере кететін жағдай парақтарың бәрі бірдей қосымша жадыға көшіріле бермейді. Жадыда әр уақытта болуы қажет парақтар қатары да бар. Мысалы, үзу механизмі көрсеткіштер массиві олардың өңделулеріне негізделген, мысалы парақ қателерін өңдеу үшін немесе кіріс/шығыс операциясын аяқтау үшін. Бұл өңделулер әр уақытта жадыда болулары керек. Олар ауыстырылмайтын парақтар белгілерімен белгіленеді.

Кейбір парақтар операциялық жүйе жұмысының барлық уақытының барысында ауыстырылмайтын болуы мүмкін, ал тағы бір түрлері тек белгілі бір уақытқа ғана ауыстырылмайтын болады. Бұндай парақтар қатарына құрамында буфер бар жады парақтары, сыртқы құрылғылармен байланысқан немесе кіріс/шығыс операцияларына

арналған парақтар жатады. Парақтар қандайда бір операция аяқталмайынша ауыстырылмайтын парақ деп белгіленеді.

3.4.

UNIX- ЖӘНЕ WINDOWS-ЖҮЙЕЛЕРІНДЕ ЖАДЫНЫ БАСҚАРУ МЕХАНИЗМДЕРІ

UNIX тобының операциялық жүйелердің алғашқы нұсқасы 1970 жылдарда пайда болды. Бұл операциялық жүйелер 16-разрядтық архитектурасы бар және жадысы 64 Кбайт болатын PDP-11 класс машиналарына қызмет көрсетуге арналған. Есептеуіш ресурстардағы шектеулер әсерінен UNIX ОЖ өнертапқыштарына жадыны басқаруға бағдарламалық оверлей тәсілін пайдалануға тура келді. Оверлей пайдаланылмай тұрған жадыны өшіру есебінен жадыны бірнеше қайтара пайдалануға және оған жаңа блоктар, кодтар және деректерді жүктеуге мүмкіндік береді. Бұл жүйе жұмысы кезінде мысалы, операциялық жүйені жүктеу және инициалдау деңгейінде орындалған және жүйе жұмысы кезінде қажеттілігі жоқ кодты өшіруге және босаған жадыны басқа бағдарламаларға қолдануға мүмкіндік берді.

Бұл тәсіл БҚ жасау кезінде қосымша талаптарды қосады, бағдарлама жасаушыдан кодтың қай бөліктерін және қашан алып тастауға немесе негізгі жадыға жүктеу керектігі туралы ереже қажет. Бұндай бағдарламалардың төзімі өте нашар, өйткені оларда көбіне өздері жасалған жүйелердің аппараттық ерекшеліктері қолданылды.

Ұрпағы операциялық жүйелерінің жадыларын басқару жүйесінің келесі даму деңгейі бағдарламаны толықтай қосымша жадыға көшіру негізіндегі стратегияны енгізу болды. Бұл жүйені қолданғанда бағдарламалар оперативті жадыға кезегімен жүктелді. Егер кезекті процеске жады жеткіліксіз екені анықталса, операциялық жүйе

процестердің біреуін негізгі жадыдан шығарып тастауға арналған арнайы бөлімге орналастырады. Процес басталған кезде бұл бөлімде қажет кеңістік резервтеліп тұрды, ол процесті негізгі жадыдан қосымша жадыға ауыстыруға қажет болатын орынның бар екеніне кепілдік болды.

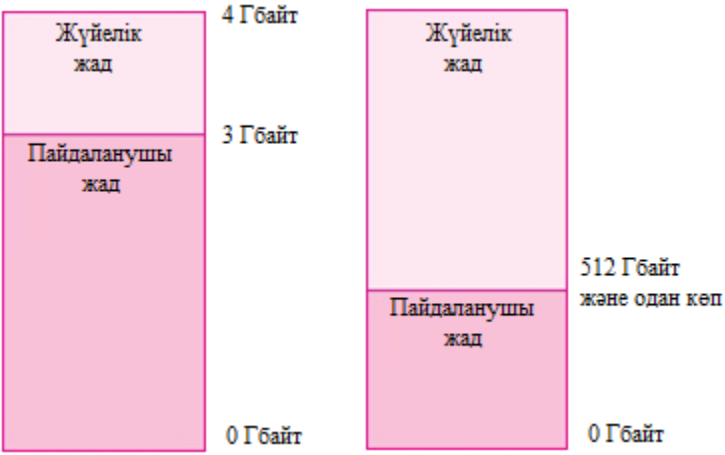
1970-ж. соңында VAX-11 класс машиналары шыққаннан кейін, UNIX операциялық жүйелері өтінім бойынша жады парақшасын жүктеу негізінде жадыны басқару тәсілін енгізуді бастады. Салдарында бұл тәсіл процестерді толық шығарып тастауға негізделген тәсілді толықтай ығыстырды. Кейбір орындалуларда жадының сегменттік құрылымына негізделген тәсіл қолданылады, бірақ UNIX жүйелерінің негізгі орындалуларында бұл тәсіл сіңісіп кетпеді және одан әрі барлық орындалулар жадыны басқарудың парақтық тәсілін қолданды.

Жадыны басқарудың парақтық тәсілі Linux тегінің операциялық жүйелерінде де қолданылады. 32-биттік нұсқаларында жадының тек 4 Гбайт қолжетімді. Осы 4 Гбайттан стандартты жүктеу кезінде 3 пайдаланушы жадысы ретінде жұмсалады, 1 Гбайт жүйелік жадыға арналады. Операциялық жүйелердің 64-биттік нұсқасын пайдалану кезінде, бұл шектеулер қолдаушы жадына 512 Гбайт дейін және одан да көп көбейеді, ал қалған адрестік кеңістік жүйелік жадыға бөлінеді (сурет-3.2).

Linux операциялық жүйелерінде UNIX тобының операциялық нұсқаларының көптеген нұсқаларындағыдай қосымша жады ретінде арнайы дисктік бөлім пайдаланылады.

Негізгі жадыдан қосымша жадыға қандай парақ көшірілетінін анықтау үшін Linux жүйесінде LRU алгоритмі қолданылады, яғни басқа парақтармен салыстырғанда ұзақ уақыт бойы пайдаланылмаған парақ ауысады. Оның үстіне көшірілетін параққа ол орындалып жатқан файлдың немесе деректері бар файлдың бөлігі екені белгілі болса және ол модификацияланбаса мұндай парақты қосымша жадыға көшірмей-ақ мүлдем өшіруге болады. Егер нәтижесінде бұл парақ қажет болса сәйкес файлдан тікелей оқылады. Егер параққа деректер модификацияланса, онда қандай да бір процеске немесе операциялық жүйеге қажет болғанша парақ күштеліп қосымша параққа ауыстырылады.

Windows операциялық жүйелерінде виртуалды жадты пайдаланатын жүйелер классына жатады. Барлық процестер виртуал жадымен жұмыс жасайды, тек жүйе ядросы физикалық жадының тікелей адресациясын пайдаланады. Жады менеджері процес пайдаланатын виртуалды адрестерді физикалыққа бейнелеу үшін парақтар кестесін қолдайды.



Сурет-3.2. Linux жүйелерінде жадыны бөлу 32- (сол жақта) және 64-бит (оң жақ)

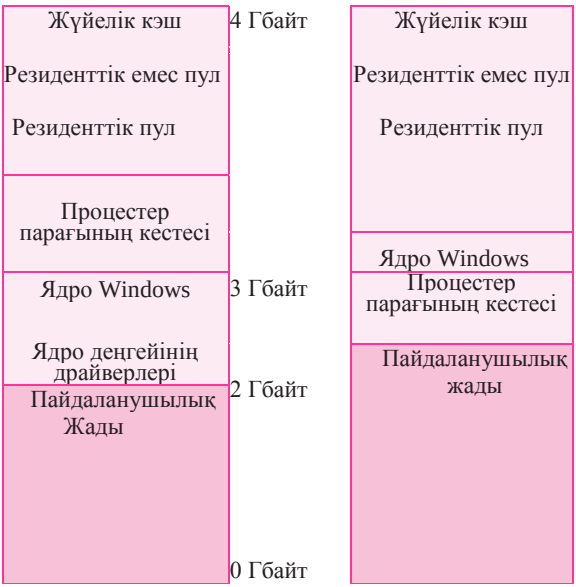
Windows операциялық жүйесінің 32-биттік нұсқасы басқаратын орындалатын процес көлемі 4 Гбайт виртуал жады кеңістігіне қолжетімділікке ие болады. жүйеге орнатылған оперативті жадының көлеміне қарамастан бұл шектеулік әрқашан сақталады. Виртуал жады екі бөлікке бөлінеді: пайдаланушыға арналған және жүйелік жады.

Пайдаланушының жады виртуалдық жадтың бөлігі, оны

операциялық жүйе қолдаушылық процестерді жүктеу, осы процестер мәліметтерін және пайдаланушы кітапханасын жүктеу үшін қолданады. Әр процестің өзінің мән мәтіні бар, осы процес пайдаланатын деректер мен код. Процес орындалғанша, жұмыс жиынтығы деп аталатын процес контекстінің бөлігі әрқашан жадыда болады.

Жүйелік жады — операциялық жүйе коды және ядро деңгейінің драйверлері орналасатын адрестік кеңістік бөлігі. Бұл жады тек ядро деңгейінде орындалатын код үшін қолжетімді, ал пайдаланушылық процестер бұл жадының бөлігіне қолжетімділігі жоқ. Бұндай бөлу барлық жүйенің тұтастығын сақтайды және операциялық жүйенің маңызды бөліктерін пайдаланушылық процестердің абайсызда жасалуы мүмкін ықпалынан қорғайды. Ядро деңгейінің драйверлері операциялық жүйенің сенімді модульдері болып саналады, және олар пайдаланушылық және жүйелік жадыға қолжетімділікке ие бола алады.

Операциялық жүйелердің 32-биттік нұсқасы виртуалды жадыдан 2 Гбайт кіші адрестік жады пайдаланушылық жадыға жатады, ал 2 Гбайт үлкендері — жүйелік. Жүйе әкімшісінде бұл шектеулерді Boot.ini. файлында орналасқан арнайы кілттің /3GB көмегімен өзгерту мүмкіндігі бар. Бұл жағдайда пайдаланушылық жадыға 3 Гбайт кіші адрестік жадылар, ал жүйелікке тек 1 Гбайт бөлінеді (сурет-3.3).



Сурет-3.3. Қалыпты жағдайда Windows 32-битті жүйесінің жадын бөлу (сол жақта) және кілтпен /3GB (оң жақ)

Windows операциялық жүйесінің 64-биттік нұсқаулығының виртуалдық адрестігінің кеңістігінің көлемі 16 Тбайт, оның ішінде 8 Тбайт пайдаланушылық жадыға беріледі және дәл сонша 8 Тбайт — жүйелік жадыға беріледі. Негізгі секциялардың орналасуы 32-биттік нұсқасымен бірдей. Секция көлемдері операциялық жүйелер нұсқаларына тәуелді және өзгере алады.

Windows операциялық жүйелердің виртуалды жадының қолжетімді көлемін арттыру үшін дисктік жинақтауыштар бөлімінде орналасатын қосымша жады қолданылады. Ол үшін Windows жады парақшаларын *тарту үшін* pagefile.sys. арнайы файл бар. Windows виртуал жадыны парақтық ұйымдастыруды қолдайды. Әдетте парақ өлшемі 4 Кбайт құрайды, бірақ Itanium процессорлары мен серверлік жүйелерде 8 Кбайт өлшемді парақтар қолданылады. Қалыпты жағдайда бұл файл бөлімнің операциялық жүйе орнатылған түбірлік каталогында орналасады. Сонымен қатар әкімші бұл файлды тарту үшін кез-келген бөлімдегі бос орынды пайдалана алады.

Қалыпты жағдайда *тартатын* файлды оның көлемі динамикалық өзгеретіндей етіп күйге келтіріле алады. Динамикалық өзгеріс пайдаланылатын виртуал жадының көлеміне байланысты. Егер тарту файлы көлемінің өзгерісі өте жиі орын алатын болса, ол өте қатты фрагменттелетін болады. Ол процес өз кезегінде тарту файлымен операция жылдамдығын өте төмендетеді және түгелімен жүйенің өнімділігі әлсірейді. Сол себепті *тартатын* файлдың көлемі алдын-ала белгіленген болатындай күйге келтіруге кеңес беріледі. Сонымен қатар тарту файлының өлшемін оперативті жады жүйесінде бекітілген көлемнен 2,5 есе көп етіп алу қажет.

Интернет-дереккөздер

https://secure.wikimedia.org/wikipedia/en/wiki/Operating_system
https://secure.wikimedia.org/wikipedia/en/wiki/Virtual_memory
https://secure.wikimedia.org/wikipedia/en/wiki/Memory_management
http://tldp.org/LDP/tlk/mm/memory.html
http://stargazer.bridgeport.edu/sed/projects/cs503/Spring_2001/kode/os/memory.htm
https://secure.wikimedia.org/wikibooks/en/wiki/Operating_System_Design/Memory_Management
http://www.technologyuk.net/computing/operating_systems/memory_management.shtml
http://msdn.microsoft.com/en-us/windows/hardware/gg487427
http://download.microsoft.com/download/7/E/7/7E7662CF-CBEA-470B-A97E-CE7CE0D98DC2/mmwin7.pptxhttp://members.shaw.ca/bsanders/WindowsGeneralWeb/RAMVirtualMemoryPageFileEtc.htm
https://secure.wikimedia.org/wikipedia/en/wiki/Paging# Windows NT

БАҚЫЛАУ СҰРАҚТАРЫ

1. Операциялық жүйелерде жадының қандай үлгілері қолданылады?
2. Виртуал жадының физикалық жадыдан ерекшелігі қандай?
3. Жадының парақтық үлгісі дегеніміз не?
4. Сыртқы жинақтауыштарда виртуалды жады бейнелеуі қай жерде орналасады?
5. Әртүрлі Windows-жүйелерінде жады қалай бөлінеді?

Тарау 4

ПРОЦЕСТЕР

4.1. ЖАЛПЫ ТҮСІНІК

Жалпы жағдайда бағдарлама — файл түрінде берілген процессор нұсқаулығының жинағы. Бағдарлама орындалуға жіберілуі үшін ОЖ алдымен тапсырманы орындайтын құрамында ресурс жады, жүйеге кіру/шығу қолжетімділік мүмкіндігін және т.б. бар орта немесе қоршау дайындауы керек. Құрамында орындалатын бағдарламаның деректері мен коды бар жады аумағы мен ортасының жиынтығы *процес* деп аталады. Процес жұмыс барысында бірнеше күйге түсе алады, әрқайсысында ол операциялық ресурс беретін ресурстарды пайдаланады.

Процестің екі негізгі күйі бар: өзінің бағдарламалық коды орындалған кезде процесті тапсырма режимінде орындау, және операциялық жүйенің ядроның адрестік кеңістігінде орналасқан ядро режимінде орындалу.

Процестерді басқару үшін операциялық жүйе процестің орындалуының барлық уақыт бойы болған жүйелік деректерді пайдаланады.

Бұл құрамдас бөліктердің барлық үйлесімділігі *процес* мән мәтін туындатады. Контекст нақ осы уақытта процестің күйін анықтайды.

ОЖ ядросы құрылымының тұрғысынан процес контексті келесі құрауыштардан тұрады [19]:

- Пайдаланушылық мән мәтін — жадының ішіндегі ақпараттар, деректер, стек, бөлінетін жады, кіріс/шығыс буферлерінің процес коды;
- регистрлік контекст — аппараттық регистрлер құраушылары (командалар есептеуіш регистрі, процессор күйінің регистрі, стекті көрсету регистрі және жалпы пайдалану регистрлері);
- жүйелік деңгей контексті — процесті сипаттайтын ядро деректерінің құрылымы. Жүйелік контекст статикалық және

динамикалық бөліктерден тұрады.

- статикалық бөліміне процес дескрипторы мен пайдаланушы алаңы (U-алаң) кіреді.

Процес дескрипторының құрамына процес идентификациясы үшін операциялық жүйе пайдаланатын жүйелік деректер. Бұл деректер қазіргі ағымдағы уақытта орындалып жатқан барлық процестер туралы ақпараттардан тұратын процестер кестесін құрастыруда қолданылады. Процес дескрипторы келесідей ақпараттардан тұрады:

- орналасу және процес алатын жады көлемі — әдетте негізгі адрес түрінде көрсетіледі. Егер процес жадыда бірнеше фрагментпен иемденсе бастапқы процестің жадыда үздіксіз таралу кезіндегі көлемін немесе бастапқы адрестер тізімі және жады блоктарының өлшемдері түрінде көрсетіледі.
- процес идентификаторы PID (Process IDentifier) — процеске ол пайда болған кезде меншіктелетін әдетте 1 бастап 65 535 аралығында болатын әмбебап бүтін сан;
- процес идентификаторы PPID (Parent Process IDentifier) — деректерден пайда болған процес идентификаторы. UNIX-жүйелеріндегі барлық процестер басқа процестерден пайда болады (мысалы, командалық интерпретатордан орындалуға бағдарлама жіберілген кезде оның процесі командалық интерпретатор процесінен пайда болды деп есептеледі);
- процес артықшылығы — процес пайдалануы мүмкін процессорлық уақыттың қатыстық шамасын анықтайтын сан. Басқару артықшылығы жоғары процеспен көбірек беріледі;
- процесті бастаған пайдаланушы және топтың нақты идентификаторы.

U-аймақ келесі ақпараттан тұрады:

- процес дескрипторын нұсқағыш;
- пайдаланушы және ядро режимінде процес орындалған уақыт есептеуіші (яғни процессорлық уақытты пайдаланды);
- соңғы жүйелік шақырту параметрлері;
- соңғы жүйелік шақырту нәтижелері;
- ашық файлдар дескрипторларының кестесі;
- процес алып жатқан адрестік кеңістіктің максимал көлемі;
- процес жасай алатын файлдардың максимал өлшемі.

Жүйелік деңгей контекстінің динамикалық бөлігі — ядро режимінде орындалған кезде процес пайдаланатын бір немесе бірнеше стектер.

Процестер кестесін қарау үшін ps команданы қолданыла алады. Командалық жолақ параметрлерінсіз орындалып ол ағымдағы пайдаланушы жіберген барлық процестерді шығарады.

Тәжірибелік қолдану үшін процестер кестесі туралы толық ақпаратты aux параметрлері бар ps шақырту арқылы алуға болады; оған қоса барлық пайдаланушылардың (a) процестері туралы ақпараттар шығарылады, дескрипторға және процес контекстіне кіретін мәліметтер бөлігі (u), сонымен қатар терминал анықталмаған процестер шығады (x):

```
$ ps aux
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.0  0.0   1324   388 ?        S    Jul06   0:25 init
[3]
root         2  0.0  0.0      0     0 ?        SW   Jul06   0:00
[eventd]
lp         594  0.0  0.0   2512   268 ?        S    Jul06   0:00 lpd
nick       891  0.1  0.1   2341   562 /dev/tty1 S    Jul06   0:18 bash
```

Жоғарыда келтірілген мысалда ps aux командаңы процестер туралы келесі ақпаратты шығарады: процесті іске қосқан пайдаланушы аты (USER), процес идентификаторы (PID), нақ осы уақытта процес алып жатқан процессорлық уақыттың пайызы (%CPU), бос емес жад пайызы (%MEM), процес алып жатқан жадының жалпы көлемі, килобайт (VSZ), әруақытта процес алып жатқан және тек процес аяқталғанда ғана босайтын жады көлемі (RSS), терминал файлы (TTY), процес күйі (STAT), процестің старт уақыты (START), қолданылу үстіндегі процессорлық уақыт саны (TIME), бағдарламаны іске қосудың толық жолағы (COMMAND).

Жұмыс барысында процеске ОЖ ресурстарына — оперативті жадыға, файлға, процессорлық уақытқа қолжетімділік ашылады.

Процестер мен қолжетімділікті басқару арасында ресурстарды үлестіру үшін ресурстарға операциялық жүйе ядросының құрамына кіретін міндеттерді жоспарлау қолданылады, сондай-ақ жадыны қорғау мен файлдар мен құрылғыларды бұғаттау механизмі қолданылады.

Міндетті жоспарлаудың негізгі атқаратын қызметі — жүйеге түсетін жүктемені процестер арасында баланста ұстау, процестердің артықшылықтарына байланысты процессорлық уақытты бөлу.

Жадыны қорғау механизмі басқа процестермен бос емес оперативті жадыға процес қолжетімділігіне рұқсат етпейді (ортақ жадыны пайдаланатын процес аралық байланыстан басқа).

Файлдар мен құрылғыларды бұғаттау механизмі бірегей қолжетімділік қағидасы бойынша жұмыс жасайды — егер қандай да бір процес жазбаға арнап файл ашса, осы файлға басқа процестер жаза алмайтындай бұғаттау қойылады.

4.2.

ПРОЦЕСТІҢ ПАЙДА БОЛУЫ. ҚАСИЕТТЕРДІ ИЕМДЕНУ

UNIX ОЖ жаңа процесті іске қосу тек орындалу үстіндегі басқа процес арқылы ғана мүмкін. Іске қосылған процес процесс-буын деп, ал іске қосқан процес —аталық процес деп аталады. Процес-буын аталық процес туралы ақпаратты өзінің дескрипторында сақтайды. Ата-ана процесі жоқ жалғыз процес — бұл 1 тең PID бар init операциялық жүйесі ядросының басты процесі. Бұл процестің іске қосылуы операциялық жүйені бастапқы жүктегенде орындалады.

UNIX ОЖ процес жасау және жаңа бағдарламаларды іске қосу механизмдері бар. Ол үшін аналықтың толық көшірмесі болып табылатын жаңа процесті жасайтын fork() жүйелік шақыртылу қолданылады:

```
#include <sys/types.h>
#include <unistd.h> pid_t fork(void);
```

Процес-буыны *және* аталық процес арасында келесідей айырмашылықтар бар:

- буын PID бірегей идентификатор беріледі, аталықтан ерекше;
- PPID процес-буынының мәні аталық процес PID мәніне көшеді;
- Процес-буыны өзінің жеке файлдық дескрипторлар кестесін алады, яғни ата-ана ашқан файлдар тобы үшін ондай файлдар қатарына жатпайды;
- Процес-буыны үшін барлық жеткізілуді күтіп отырған сигналдар тазаланады (10.3-бөлімін қараңыз);
- Процес-буынының уақыттық орындалу статистикасы ОЖ кестесінде нольге айналады;
- аталыққа орнатылған жадыны және жазбаны бұғаттау қайталанбайды.

Осыған қарамастан процес келесідей қасиеттерді аталық процестен иемденеді:

- бағдарламаның командалық жолақтарының аргументтері;
- ауысымдық орта;
- процесті іске қосқан пайдаланушылардың (UID) нақты және тиімді (EUID) идентификаторлары;
- процесті іске қосқан топтардың шын (GID) және тиімді (EGID) идентификаторлары;
- артықшылық;
- сигналдарды өңдеу құрылғылары (10.3.5-бөлімін қараңыз).

Егер *процес-буынын* жасау сәтсіздікке ұшыраса fork() қызметі -1 мәнін қайтарады. Оның себебі келесі жағдайлардың бірі болуы ықтимал:

- ағымдағы пайдаланушы үшін процестердің максимал санының асып кетуі. Мысалы BASH қабатын қолданған кезде ағымдағы шектеулерді ulimit -u команданың көмегімен білуге немесе өзгертуге болады;
- рұқсат етілген еншілес процестер санының толып кетуін getconf CHILD_MAX команданы арқылы еншілес процестердің максимал санының шектеуін білуге болады;
- жүйеде процестердің максимал санының асып кетуі. Бұл шектеуді жүйе өзі қояды және оны sysctl fs.file-max команданың көмегімен немесе /proc/sys/fs/file-max файлын қарау арқылы білуге болады.
- оперативті жады мен *картту* бөлімінің көлемімен анықталатын виртуал жады сарқылған.

Fork () атқарымы сәтті орындалған жағдайда процес-буынның PID аталық процеске және 0 тегі-процеске қайтарады. Процестер идентификаторларын сақтау үшін pid_t айналмалы түрі қолданылады. Көп жүйелерде бұл түрі int типіне ұқсас, дегенмен int типін қолдану UNIX болашақ нұсқаларымен ұқсастықты қамтамасыз ету ұсынылмайды.

PID және PPID мәндерін алу үшін сәйкесінше getpid() және getppid() атқарымлары қолданылады. Олар қайтаратын мәннің типі— pid_t:

```
#include <sys/types.h>
#include <unistd.h>

pid_t getpid(void);
pid_t getppid(void);
```

fork() атқарымы шақыртылғаннан кейін екі процестің орындалуы да — аталық және топтар олардан кейін еретін командалар арқылы жалғасады, әр процес орындайтын бағдарламалық кодты шектеу қажет. Ол үшін ең айқын тәсіл — fork() қызметін қайтаратын тексеруші кодтың қажет бөліктерін әр түрлі шарт тармақтарында орындау.

Осындай тексерістің классикалық үлгісіне айналған мысал төменде келтірілген:

```
#include <sys/types.h>
#include <unistd.h
int main(void)
{
pid_t pid;
switch (pid = fork())
{
case -1:
/* fork() Сәтсіз орындалу – pid тең -1 */
perror("Unsuccessful fork() execution\n"); break;

case 0:
/* Еншілес процес денесі */
/* pid = 0 – бұл еншілес процес. */
/* Онда pid мәні нольмен инициацияланады */ sleep(1);
printf("CHILD: Child spawned with PID = %d\n",
getpid());
printf("CHILD: Parent PID = %d\n", getppid());

/* Еншілес процес жұмысының аяқталуы */
_exit(0);

default:
/* процес денесі-аналық процесіне жатады */
/*  pid>0. Демек, тегінің pid алған аналық процес  /*
/*орындалып жатыр */
printf("PARENT:Child spawned with PID=%d\n",
pid);
printf("PARENT:Parent PID=%d\n", getpid());
}

/*   Процес   денесі-fork()өңделгеннен   кейін   аналық
процесінің денесі */
/* Егер case 0 _exit(0) көрсетілмесе, онда*/
/* тегінің артынан ерген команданы орындаған болар еді*/

exit(0);
}
```

Бағдарлама келесі жолақтарды шығарады (процестер идентификаторлары бір-бірінен ерекшеленуі ықтимал):

```
PARENT:Child spawned with PID=2380
PARENT:Parent PID=2368 CHILD: Child
spawned with PID = 2380 CHILD: Parent
```

```
PID = 1
```

Бағдарламаның келтірілген мысалында келесі іске назар аудару қажет: егер case сәйкес нұсқа ішінде топ-процесінен шығуды қамтамасыз етпесе, онда топ case ішінде бағдарламалық кодты орындауды аяқтап болған соң, switch() құрылысының жабылатын жақшасынан кейін орналасқан кодты орындауды жалғастырады. Көп жағдайда бұндай әрекеттің алдын алу керек. Берілген қайтару кодымен процестен нақты шығу үшін _exit(<қайтару коды>) атқарымы қолданылады. Бұл атқарым шақырту процесін жойып, келесі әрекеттердің орындалуына алып келеді:

- жеткізілуді күтіп отырған барлық сигналдар өшеді (10.3-бөлімін қараңыз);
- барлық ашық файлдар жабылады;
- ресурстарды пайдалану статистикасы proc файлдық жүйеде сақталады;
- аталық-процес хабарлайды және топтардағы PPID орын ауыстырылып қойылады;
- процес күйі «*зомби*» ауысады (4.3-бөлімін қараңыз). Процес жұмысын аяқтаудың басқа тәсілдері басқа бөлімдерде қарастырылатын болады.

процес-*топты* аяқтау үшін exit() атқарымының орнына _exit() қолдану керек. Ал, _exit() атқарымы процеске қатысты ядро құрылымын — дескриптор мен процес контексті тазартады. Одан ерекше exit() атқарымы көрсетілген әрекеттерге қосымша пайдаланушы деректерінің барлық құрылымын бастапқы күйге қайта әкеледі. Мұның нәтижесінде аналық-процестің құрылымына зақым келетін жағдай туындауы ықтимал. Exit () қызметі тек аналық-процеспен орындалатын атқарымдарда пайдаланылады.

Көп жағдайда fork() атқарымы exec...() тобының атқарымдарының бірімен бірге қолданылады. Бұл атқарымдарды бірге қолдану кезінде жаңа процесті жүргізу және одан жаңа бағдарламаны жіберу мүмкіндігі бар.

Exec...() атқарымын орындау нәтижесінде бағдарламалық код пен процестердің деректері жіберіліп жатқан бағдарламаның бағдарламалық кодына ауысады. Өзгеріссіз қалатын тек процес идентификаторы PID мен PPID аналық процес идентификаторы.

Егер exec...() тобының атқарымының орындалуы сәтсіз аяқталса, ол мынадай себептерден болуы ықтимал:

1. Орындалатын файлға апаратын жол жүйелік параметр PATH_MAX мәнінен асып түседі, файлға дейінгі жолдың элементі NAME_MAX жүйелік параметр мәнінен асып түседі немесе файл атауының рұқсаты кезінде жолдағы символдық сілтемелер саны

SYMLOOP_MAX санынан асып түскені белгілі болды. Бұл параметрлердің мәндерін getconf -a командаңы көмегімен анықтауға болады.

2. Атқарым шақырылатын файл орындалмайтын болып саналады, қарапайым файл емес немесе ондай файл мүлдем жоқ.

3. Іске қосылып жатқан процеске берілетін параметрлер тізімі өте ұзақ.

Параметрлердің максимал рұқсат етілген саны максимал рұқсат етілген командалық жолақ ұзақтығы негізінде анықталады. POSIX стандартына сай бұл мән 4 096 аз болмау керек. Нақты жүйелерде бұл мән бірнеше есе үлкен. Жүйе үшін бұл шектеуді анықтау үшін getconf ARG_MAX командаңы қолдануға болады. Параметр ARG_MAX командалық жолақта рұқсат етілген символдардың максимал санынан тұрады. Шынында орындалатын командаңың максимал ұзындығына қосымша шектеулер қойылады. POSIX стандартына сай орындалу үстіндегі командаңың максимал рұқсат етілген ұзақтығын expr 'getconf ARG_MAX' - 'env|wc -c' - 2048 командаңың көмегімен есептеп шығаруға болады. Оған қоса жүйе бір параметр ұзындығына да шектеу қояды. Төменде келтірілген мысалды аталық-процес жаңа процесті туындатады, ол параметрі -1 бағдарламасын /bin/ls іске қосады. Топ-процесінің кодын толықтай ауыстыру жүріп жатқандықтан _exit()т атқарымын шақыру міндетті емес:

```
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>

int main(void)
{
pid_t pid; int status; if (fork() == 0)
{

/* топтың денесі */ execl("/bin/ls", "/bin/ls", "-l",
0);
}
/* аналық дененің жалғасы */ wait(&status);
printf("Child return code %d\n",
WEXITSTATUS(status));
return 0;
}
```

Бағдарлама келесіні шығарады:

```
total 17
```

```
-rwxr-xr-x 1 nick users 9763 Oct 11 15:41 a.out -
rwxrwxrwx 1 nick users 986 Oct 11 15:20 fork1.c -
rwxrwxrwx 1 nick users 321 Oct 11 15:40 fork2.c Child
return code 0
```

Аталық-процес топ жасағаннан кейін оның аяқталуын wait() атқарымы көмегімен күтеді:

```
#include <sys/types.h>
#include <sys/wait.h>
pid_t wait(int *status);
```

Бұл атқарым топ-процесінің орындалып болуын (егер олар бірнеше болса, олардың қайсысы бірінші аяқталатыны маңызды) Атқарым қайтаратын мәнді — аяқтайтын процестің PID мәнін күтеді. Статус мәнін сілтеме бойынша беретін атқарымдар топтың қайтып келу коды туралы ақпаратты кодтайтын санды және аяқталу сәтіндегі оның күйін білдіреді. Қызықтыратын ақпаратты қарау үшін WEXIT...0 макроанықтауыштардың бірін пайдалану қажет. Мысалы, WEXISTATUS() макроанықтауыш статус мәніндегі қайтару кодының номерін қайтарады.

Егер процес бірнеше топты іске қосып нақты бір процестің аяқталуын күту қажет болса waitpid() атқарымы қолданылады:

```
#include <sys/types.h>
#include <sys/wait.h>
pid_t waitpid(pid_t pid, int *status, int options);
```

бұл атқарымның бірінші аргументі — аяқталуын күтіп отырған PID процесі, екіншісі – статус мәні. Үшінші параметр атқарымның жұмыс режимін анықтайды.

Егер үшінші параметр 0 тең болса, ең болмағанда бір топ аяқталмайынша процестің орындалуы тоқтайды.

Егер үшінші параметр ретінде WNOHANG константа берілсе, статус мәні тек топ өз орындалуын аяқтағаннан кейін ғана меншіктеледі; ол орындалмаған жағдайда аналық процестің орындалуы жалғаса береді. Егер WNOHANG параметрі шықса және топ әлі аяқталмаса, онда waitpid() атқарымы 0 қайтарады:

```
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>

void main(void)
{
```

```
pid_t childPID;
pid_t retPID = 0;
int status;
if ( (childPID = fork()) == 0)
{
/* Топ денесі */
execl("/bin/ls", "/bin/ls", "-l", 0);
}
while (!retPID) /* Аталық денесінің жалғасы */
{
retPID = waitpid(childPID, &status, WNOHANG);
}
printf("Child return code %d", WEXITSTATUS(status));
}
```

Бағдарлама келесі ақпаратты шығарады

```
:total 17
-rwxr-xr-x 1 nick users 9763 Oct 11 15:41 a.out -
rwxrwxrwx 1 nick users 986 Oct 11 15:20 fork1.c -
rwxrwxrwx 1 nick users 321 Oct 11 15:40 fork2.c
Child return code 0
```

4.3.

ПРОЦЕСТІҢ КҮЙІ. ПРОЦЕСТІҢ ӨМІРЛІК АЙНАЛЫМЫ

Процестің жасалуы және аяқталуының уақыты арасында процес операциялық жүйеден бірнеше оқиғалардан шабуыл жасалса, соған байланысты бірнеше күйге түседі.

fork() атқарымының көмегімен пайда болғаннан кейін бірден процес «құрылу» күйінде болады — процестер кестесінде жазба ол үшін әлдеқашан бар, дегенмен процес деректерінің ішкі құрылымы әлі инициализацияланбаған. Процестің алғашқы инициализациясы

аяқталған бойда, ол «Іске қосуға дайын» күйіне ауысады. Бұл күйде процеске барлық қажетті ресурстар қолжетімді, тек процессорлық уақыт қолжетімді емес, ол орындалуды күтіп тұрған тапсырмалар кезегінде орналасады. Жоспарлағыш процесті таңдаған бойда, ол «Ядро режимінде орындалады» күйіне ауысады, яғни операциялық жүйе ядросының процестің соңғы өзгерген күйін өңдейтін бағдарламалық кодын орындайды. Осы күйден ол «Тапсырма режимінде орындалуда» күйіне ауыса алады, ол күйге ауысқанда ол өзінің бағдарламалық кодын орындайды. Әр жүйелік шақыртуда процес «Ядро режимінде орындалуда» күйіне ауысып отырады (сурет-4.1).

Жүйелік шақыртулар белгілі бір ресурстарға рұқсат алу үшін орындалуы мүмкін, ал ресурстар қолжетімді болмаған жағдайда «Ядро режимінде орындалуда» күйінен шығып, «Күту» күйіне түседі. Ол күйде процессорлық уақытты босатады және ресурстың босауын күтеді. Ресурс қолжетімді болған кезде процес оны ұстап алады, «Іске қосуға дайын» күйіне ауысады да, тағы да процес жоспарлағышының таңдауын күтеді.

«Ядро режимінде орындалуда» және «Тапсырма режимінде орындалуда» күйлерінде жоспарлағыш басқаруды басқа процеске бере алады. Бұл ретте басқаруды алып кеткен процес «Іске қосуға дайын» күйіне ауысады.

Сурет-4.1. Процестің өмірлік айналымы

Белсенді процесті қайта қосқан кезде мән мәтіннің ауыстырылуы орындалады, сонымен бірге ядро ескі процес мән мәтінін сақтайды және басқару беріліп жатқан процестің мән мәтінін қайта қалпына келтіреді.

Аяқталғаннан кейін процес «Зомби» күйіне ауыстырылады, яғни процес алып жатқан жады босайды, ал процестер кестесінде процесті қайтару коды туралы ақпарат қалады. Процесті толық аяқтау үшін аталық-процес wait() атқарымын шақыртуы көмегімен жүзеге асады.

4.4. ТЕРМИНАЛ. ҚОРЫТЫНДЫНЫ БУФЕРЛЕУ

Қалыпты жағдайда кіріс/шығыс жүйелік кітапханасы терминал құрылғысына жүйелік шақырту орындаған бойда бірден бермейді (мысалы, printf()), жадының арнайы аймағына мәтіннің белгілі бір мөлшері жиналуына байланысты береді.

Жадының мұндай аймағы қорытынды буфері деген атау алды, ал деректерді оларды құрылғыға жіберу алдында жинау процесі буферлеу деп аталады. Әр процеске өзінің кіріс/шығыс буфері бөлінеді. Олардың деректері құрылғыға операциялық жүйенің ядросынан команда келіп түскен кезде немесе буфер толуы бойынша жіберіліп отырады. Аралық буферді пайдалану терминалдың физикалық құрылғысына жүгіну мөлшерін өте азайтуға мүмкіндік береді және жалпы онымен жұмысты жылдамдатады. Жүйелік кітапхананың буферлеуінің үш түрі бар:

1) Толық буферлеу — терминалдың физикалық құрылғысына деректерді жіберу тек буфер толық толғаннан кейін ғана орындалады;

2) Жолма жол буферлеу терминалдың физикалық құрылғысына деректерді жіберуді мәтіннің бір жолағын шығарғаннан кейін жүргізеді (яғни, жолақтың жылжуымен аяқталатын символдардың тізбегі немесе ұзындығы терминалдың еніне тең символдар тізбегі);

3) Буферлеудің болмауы, осы ретте шығару буферінде алдын ала жинақталмай деректер терминалдың физикалық құрылғысына бірден жіберіледі.

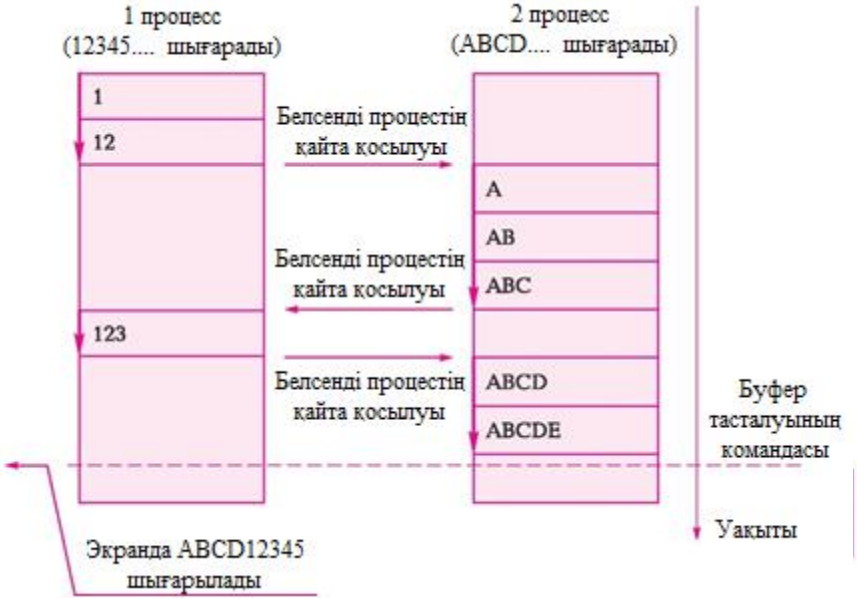
Бірнеше процестен бір мезетте бірнеше мәтіннің терминалға шығарылуы алдында, процестер шығарған мәтіндер әр процестің бөлек буферінде жинақталады, ал терминалға деректерді жіберу сәті буфердің ішіндегісін құрылғыға тастау туралы команданы ОЖ жіберген кезде орындалады. Әртүрлі процестер шығарған деректердің бірдей емес ұзындығы және қосулы буферлеу кезінде оларға процесорлық уақытты бірдей бөлмеу нәтижесінде, параллель орындалатын процестердің шығарылуы емін-еркін араласып кетуі ғажап емес. Деректердің қалай араласып кететіні процестер буферінен терминал құрылғысына деректерді беру уақытына тәуелді.

Мысалы, 4.2-суретте уақыт сызығында екі процестің кіріс/шығыс буферлер күйінің тізбектей өзгерісі көрсетілген, экранға тізбектей шығарылған араб сандарымен 1 ден 9 дейін және A бастап F дейін латын әріптері.

Бұл процестерде әр символды шығару printfQ бөлек операторы арқылы жүргізіледі. Суретте қою нұсқармен процес белсенді кезіндегі уақыт көрсетілген (процессорлық уақытты қолданады), әр тікбұрыштың биіктігі процес ұзақтығының жалпы уақытын көрсетеді.

Буферлерде деректердің жинақталуы баяу жүретіндіктен және printf() атқарымын бөлек шақыртуы арасында белсенді процестің ауысуы орындауы мүмкін болғандықтан, деректер буферлерде біркелкі жиналмайды.

Буферді тастау командасын келген кезде экранға шығарылатын мәтін көлемі буферде қандай көлем жиналғанына байланысты, ал мәтін



фрагменттерін экранға шығару тізбегі буферлерді экранға тастау тізбегімен анықталады.

Сурет-4.2. Параллель орындалатын процестердің қосу/шығару буферлерін тастау

Бұл мәселенің алдын алу үшін буферлеуді сөндіріп тастауға болады. Бұл деректерді шығару командасын орындау мен терминалда мәтіннің нақты шығуы арасында кешігуді минимизациялайды.

Буферлеуді басқару setvbuf() атқарымы арқылы жүргізіледі:

```
#include <stdio.h>
int setvbuf (FILE *stream, char *buf, int type,
             size_t size);
```

Аргумент stream буферлеу режимі өзгеретін қосу/шығару ағынының атауын береді; buf буфер сақталатын жады аймағына нұсқарды белгілейді; type буферлеу түрін анықтайды және келесідей мәндерді қабылдай алады:

- _IOFBF — толық буферлеу;
- _IOLBF — жолма-жол буферлеу;
- _IONBF — буферлеудің болмауы.

Size аргументі buf нұсқасы сілтенетін буфер өлшемін береді.

Шығарудың стандартты ағынына буферлеуді өшіріп тастау үшін нольдік өлшемге ие буферді пайдаланып және төмендегідей setvbuf() атқарымын шақыруға болады:

```
setvbuf(stdout,(char*)NULL,_IONBF,0);
```

Бұл атқарымды деректерді бір терминалға шығаратын процестерді туындататын кез-келген бағдарламаның жұмысы басталған кезде шақыруға кеңес беріледі.

Бұдан басқа дәл setvbuf() атқарымы секілді, осы атқарым секілді қызмет ететін setbuf() атқарымы бар, ол қажет болса буферлеу типін өзгертпестен буферді сөндіріп тастайды немесе жаңа буферді жасай алады. Бұл атқарымның интерфейсі келесідей:

```
#include <stdio.h>
void setbuf(FILE *stream, char *buf);
```

Сәйкесінше буферлеуді келесі команда арқылы сөндіріп тастауға болады:

```
setbuf(stdout,(char*)NULL);
```

БАҚЫЛАУ СҰРАҚТАРЫ

1. Операциялық жүйеде процесті қандай параметрлер сипаттайды?
2. Процес өзінің өмір айналымы барысында қандай күйлерден өтеді?
3. Жаңа процесті қандай команда жасайды?
4. fork() атқарымын орындаған кезде не болады?
5. wait() жүйелік шақыртуы қандай роль атқарады?
6. Өз процесінің номеріне сәйкес қайтып келу кодымен аяқталатын өзінің 10 тобын туындататын бағдарламаны жазыңыз.

Тарау 5

ТАПСЫРМА

5.1.

ТАПСЫРМАЛАРДЫ БАСҚАРУ ТІЛДЕРІ

Командалық интерпретатор — пайдаланушының ОЖ барлық жұмыс сеансында жүргізілген бағдарлама. Оның негізгі қызметі — берілген командалық интерпретатор тілінде жазылған пайдаланушының командасын орындау және бұл командаларды тікелей (интерпретаторға кіріктірілген құралдармен) немесе басқа бағдарламаларды шақыру арқылы орындау.

Командалық интерпретатордың пайдаланушымен байланыс жасауының негізгі тәсілі командалық жолақ интерфейсі болып табылады. Бұлай қатынас жасау тәсілінде жүйемен жұмыс жасауға арналған негізгі құрал ретінде *терминал* қолданылады. Терминал ақпаратты шығару және пайдаланушы ақпаратты енгізу үшін қызмет етеді. Физикалық терминал— бұл монитор және пернетақта. Логикалық, операциялық жүйе тұрғысынан терминал — бұл екі файлдан құралған жинақ. Бұл файлдың біреуі ақпаратты енгізу үшін пайдаланылады (пернетақтадан келетін ақпарат), екіншісі – ақпаратты шығаруға арналады (экранға шығарылатын ақпарат).

Кейбір операциялық жүйелерде терминал жазба үшін де ақпаратты оқу үшін де пайдаланылатын бір файлдан тұрады.

Осы ретте физикалық құрылғы файлмен жұмыс жасау режимімен анықталады — файлға жазылатын деректер экранға шығады, пернетақтадан енгізу оқылатын файлдағы ақпаратты толықтыра түседі.

Жұмыс басталуы үшін сигнал ретінде экранға шығатын *командалық жолақтың шақыртуы* немесе жай ғана *шақырту* қызмет етеді — интерпретатор команданы енгізуді күтіп тұрғанын көрсететін символдар тізбегі.

Команданы енгізу [Enter] клавишін басумен аяқталады, содан кейін интерпретатор команданы орындауды бастайды. Мысалы, терминалға сәйкес келетін файл атауын шығаруға болады. Ол tty команданы көмегімен орындалады:

```
$ tty <Enter>
/dev/console
$
```

Бұл жерде /dev/console — деректерді шығаруға және пайдаланушы енгізген деректерді оқуға қолданылатын файл атауы.

Пайдаланушының командалары терминалдан диалогтық режимге (командалық жолақ интерфейсі арқылы) немесе пайдаланушының қатысуынсыз пакеттік режимге беріле алады. Пакеттік режимде жұмыс жасаған кезде командалар тізбегі мәтіндік файл түрінде рәсімделеді. Командалардың тізбегі командалық интерпретатормен орындалатын тапсырманы анықтайды. Сол үшін командалық интерпретатор тілдерін тапсырмаларды басқару тілі деп те атайды. Тапсырманы анықтайтын файл көбіне сценарий деп аталады.

Тапсырмаларды басқару тілі келесі қасиеттерге ие болуы тиіс:

- Тапсырмалардағы бағдарламаларды орындау тізімін анықтайтын құралдары және қолданылатын ресурстарды анықтау құралы болуы керек;
- Бағдарламаны орындау типін анықтайтын құралдар болуы керек (негізгі режим, фондық режим, артықшылықты анықтау және т.б.);
- тапсырма бөлігінің орындалу шартын және тапсырманың тармақтану шарттарын орындау құралдары болуы тиіс;
- ОЖ (файлдар мен процестер) ресурстар күйін және олардың қасиеттерін (мысалы, қолжетімділік құқығы) тексеру құралы болуы керек.

5.2. ПАКЕТТІК ӨҢДЕУ

Тапсырмамен жұмыс екі режимде орындалады: диалогтық режимде және пакеттік өңдеу режимінде.

Диалогтық режимде жұмыс жасау дегеніміз тапсырманы орындау кезінде пайдаланушымен әрқашан диалог жасап отыру. Пайдаланушымен диалог жасау барысында қажеттіліктің пайда болуына байланысты тапсырмаға, бағдарлама құрамына кіретін командалар мен бағдарламалардың орындалу параметрлері

анықталады.

Егер тапсырманы орындау уақыты ұзақ болса (сағат немесе күн), онда диалогтық режиммен жұмыс жасау ақылға қонымсыз, өйткені тапсырма орындалу кезінде терминалда оператор отыру керек.

Бұл жағдайда операторға жүктемені азайтудың бір тәсілі — тапсырманы пакеттік режимде орындалатындай етіп құрастыру.

Пакеттік режимде барлық қажет деректер мен параметрлер тапсырмасы пакеттік режимде орындалар алдында дайындалады. Тапсырманың орындалуы оператордың қатысуынсыз жүргізіледі, ал барлық диагностикалық хабарламалар және қате туралы хабарламалар орындалу хаттамасы файлында жинақталады. Диагностикалық хабарламалар пакеттік режимде пайда болуына байланысты терминалға шығарылуы мүмкін, бірақ пайдаланушыға тапсырма орындалып болғаннан кейін экранға шығарылған тек соңғы хабарлама ғана қолжетімді болады. Егер диагностикалық хабарламалар саны терминал жолақтарының санынан асып кетсе, бірінші хабарламалар автоматты түрде өшіріліп қалады.

Тапсырмаға орындалмай тұрып мәліметтерді беру үшін тапсырма параметрлері қолданылады. Егер тапсырма командалық жолақ интерфейсі болып саналатын командалық интерпретатормен өңделсе, тапсырма параметрлері командалық жолақ параметрлері ретінде анықталады.

Командалық жолақ дегеніміз шақыртылатын тапсырманың атауы және оған берілетін параметрлер көрсетілген мәтіндік жолақ. Барлық параметрлер бір-бірінен бөлгіш-таңбалар арқылы бөлінеді, мысалы бос орындармен. Параметрлер — символдар тізбегінің, файлдар мен құрылғылар атауының және т.б. сандық тұрақтылар көрсететін жолақтар. Параметрлер интерпретациясы позиционды: параметрлер бөлгіштер арқылы ерекшеленеді, сол себепті бірінші және екінші параметрді анықтамай үшінші параметрді анықтауға болмайды. Берілетін параметрлердің құрамында жалпы айтқанда бөлгіш-символдардан және әртүрлі басқарушы кодтар болмау керек. Дегенмен параметр мәтінінде бос орынды қолдануға болады: оны қос жақшаға алады.

Команданы беруге болатын параметрлердің максимал саны алынған команда мен жүйенің максимал ұзындығымен анықталады. Әр түрлі жүйелерде бұл сан ауысып отырады. POSIX стандартына сай бұл мән 4 096 кем болмау керек. Жолақтың шектеуін анықтаудың ең оңай тәсілі — мына команданы орындау

```
xargs --show-limits
```

бұл команда сіздің жүйеңізге салынған шектеуді анықтауға ықпал етеді. Осылайша, «Білімді бақылау» жүйесінде give.sh тапсырмасы бар, ол белгілі бір студентке өткен тақырып бойынша бақылау жұмысының нұсқаларын таратуға арналған. Осы тапсырманы шақыртуға арналған қарапайым командалық жолақ төменде берілген:

```
give.sh 1 5 vasya
```

Бұл жерде give.sh — тапсырма атауы; 1 — тақырып санын анықтайтын тапсырманың бірінші параметрі; 5 — нұсқа номерін анықтайтын тапсырманың екінші параметрі; vasya — тапсырма берілетін студент атын анықтайтын үшінші параметр.

Тапсырманы орындау нәтижесі тапсырманың ақпараттық ортасының, яғни файлдар мен каталогтар күйі мен құрамының өзгертілуімен, пайдаланушының басқа тапсырмалары мен бағдарламаларының іске қосылуы мен тоқтауымен сипатталады. Тапсырманың орындалғаны туралы есеп жоғарыда айтылғандай экранға шығарылады немесе файлға барады. Тапсырманың орындалу нәтижесін автоматты түрде өңдеуді жеңілдету үшін оны орындап болған соң қайтару коды қалыптасады— орындалудың сәтті шыққанын сипаттайтын сандық мән. Қайтару кодының нақты мәні тапсырма мәтінінде анықталады. Тапсырманы қайтару кодына қолжетімділік аяқталғаннан кейін, командалық интерпретатордың арнайы құралдары көмегімен тікелей алуға болады.

Осылайша, «Білімді басқару» жүйесіне look.sh тапсырмасы кіреді, оқытушының жұмыс аймағында орналасқан берілген тақырып бойынша нұсқалар санын қарауға арналған. Осы тапсырманы іске қосу үшін оған тақырып номері беріледі:

```
look.sh 3
```

тапсырма өзінің орындалу нәтижесінде тақырып бойынша берілген нұсқалар санына тең қайтып келу сандық кодын қалыптастырады да оны өзі аяқталғанда қайтарады. Бұл қайтару кодын пайдаланушы немесе басқа тапсырма көре алады.

Қазіргі таңда диалогтық режим соңғы пайдаланушы үшін әлдеқайда ыңғайлы (мысалы, Windows операциялық жүйесінің бақылауымен жіберілетін көптеген бағдарламалар дәл осы диалогтық режимде жұмыс жасайды). Дегенмен, орындалудың пакеттік режимі көптеген бұрынғы операцияларды орындауда ыңғайлы болуы мүмкін. Сол себепті барлық дерлік операциялық жүйелерде командалық

интерпретаторлар немесе оларға ұқсас тапсырмаларды пакеттік режимде орындауға мүмкіндік беретін бағдарламалық құралдар бар.

5.3. BASH ИНТЕРПРЕТАТОРЫ ТІЛІНІҢ ЖАЛПЫ ЕРЕЖЕЛЕРІ

Тапсырманы басқару тілінің синтаксисі пайдаланылатын командалық интерпретатормен анықталады. Осы оқулықта базалық ретінде командалық интерпретатор BASH (Bourne - Again Shell) қолданылады. BASH команданың диалогтық режимде жұмыс жасаған кезде командалық жолақ шақыртуына жауап пернетақтадан енгізіледі.

Файлдар түрінде рәсімделетін тапсырма екі бөліктен — командалық интерпретатор атын және оған апаратын жолды анықтайтын тақырыптан және тапсырма мәтінінен тұрады. Тақырып тапсырма файлының бірінші жолағының бірінші таңбасынан басталады және BASH интерпретаторы үшін әдетте келесідей жазылады:

```
#!/bin/bash
```

Бұл жерде «#» — түсініктеме таңбасы. Жолақтағы «#» таңбасынан кейін орналасқан барлық таңбаларын интерпретатор команда ретінде қабылдамайды, тапсырма орындаған кезде еленбейді.

Тақырыпта ерекше түрдегі түсініктеме болып саналады, өйткені «#» таңбасынан кейін бірден «!» таңбасы қойылған. «#!» конструкциясы тапсырма файлының алдына орналастырылғанда, одан кейін командалық интерпретатордың орындалатын файлының толық атауы жазылатыны туралы ескертеді.

Тақырыптан кейін сценарийдің негізгі бөлігі— командалар тізбегі жүреді. Сценарийдің бір жолағы бір немесе бірнеше командалардан, түсініктемелерден тұруы мүмкін немесе бос болуы да ықтимал. Ереже бойынша сценарийдің бір жолағы бір командан тұрады, егер командалар саны көп болса бір жолақта олар өзара үтір нүктемен бөлінеді.

BASH көптеген командаларының синтаксистік шақыртуы екі бөліктен тұрады:

```
<команда атауы> <параметрлер>
```

Команда аты ретінде BASH ішкі команданы немесе бағдарлама коды мен тапсырма мәтінінен тұратын файл атауын пайдалануға болады.

BASH жиі қолданылатын командалары және сыртқы бағдарламалар бойынша қысқаша анықтама 2 Қосымшада келтірілген.

Егер команда және оның параметрлері өте ұзын болса, алмастырып

қою таңбасын қолдануға болады «\». Осы таңбаны көрсеткеннен кейін команданы келесі жолақта жалғастырып жаза беруге болады, ал командалық интерпретатор ауыстыру таңбасынан кейінгі жазбалардың барлығын команданың жалғасы ретінде қабылдайды.

BASH командасы бойынша құжаттаманы қарау үшін кіріктірілген man анықтамалық жүйе қолдануға болады. Көмек парақшасын BASH шақырту үшін «man bash» енгізу қажет. BASH кіріктірілген командаларының сипаттамасы BUILT-IN COMMANDS бөлімінде орналасқан.

UNIX кез-келген команда бойынша анықтамалық команданы қарау үшін – команда атауы көрсетілген параметрі бар man шақырту маңызды. Командалар атауын іздеу үшін және олар туралы қысқаша ақпараттар алу үшін apropos бағдарламасын қолдануға болады. Мысалы, apropos edit шақыртуы бағдарламалар бойынша атауында edit кездесетін барлық анықтамалық парақтарды экранға шығарады.

5.4. АЙНЫМАЛЫЛАР

Тапсырмаларды жазған кезде айнымалыларды (жоғары деңгей тілдерінде бағдарламалау жасалғанға ұқсас) анықтау және пайдалану мүмкіндігі бар. Айнымалылардың екі түрі бар: ішкі айнымалылар және айнымалы орта, яғни ОЖ арнайы аймағында берілген және барлық орындалып жатқан бағдарламаларға қолжетімді айнымалылар. Айнымалылар ортасы — пайдаланушы бағдарламасының орындалуына ықпал ететін ақпараттық ортаның бөлігі.

5.4.1. Айнымалылар мәндерімен жұмыс

Тапсырманы орындау барысында мәні анықталмаған айнымалыға жүгінген кезде қате шығатын болады. Бұл BASH ішінде мәннің жоқ болуы мен бос жолақ арасында айырмашылық болуымен байланысты. Қателіктің пайда болуының алдын-алу үшін айнымалы мәндеріне қолжетімділіктің әртүрлі тәсілдері бар:

- $var — var айнымалысының мәнін немесе егер ол инициализацияланбаған болса бос мәнді алу;
- ${var} — $var барабар;
- ${var:-string} — егер анықталған болса var айнымалысын алу, немесе егер var айнымалысы анықталмаған болса string жолағын

шығару;

- ${var:=string} — егер анықталған болса var айнымалысын алу, немесе егер var айнымалысы анықталмаған болса string жолағын шығару. Онымен қоса егер айнымалысы анықталмаған болса оған string мәні меншіктеледі;
- ${var:?string} — егер анықталған болса var айнымалысын алу, немесе егер var айнымалысы анықталмаған болса string жолағын шығарылады да тапсырма аяқталады.
- ${var:+string} — егер var] анықталған болса, string мәнін алу, немесе егер var айнымалысы анықталмаған болса бос мән алынады[8].

Айнымалы мәнінің орнына string мәнін алмастырып пайдалану өте қолайлы, мысалы келесі жағдайда: тапсырма экранға файлды cat команданы арқылы шығарады деп болжайық. Шығарылатын файлдың атауы айнымалы ортада бар. Егер айнымалы орта берілмесе — файлдың алдын-ала берілген атауы алмастырылып қойылады:

```
cat ${FILENAME:-/home/sergey/default.txt}
```

Осылайша, егер айнымалы FILENAME анықталмаған болса, cat команданы экранға шығарады:

```
/home/sergey/default.txt.
```

5.4.2. Жүйелік айнымалылар

Пайдаланушы анықтайтын айнымалыдан басқа BASH командалық интерпретатордың кіріктірілген жүйелік айнымалылары бар. Бұл айнымалылардың стандартты атаулары және белгіленген түсіндірмесі бар, сонымен бірге олардың мәндері командалық интерпретатормен беріледі, пайдаланушы оны тек оқи алады, бірақ өзгерістер енгізуге болмайды.

BASH ішінде келесідей кіріктірілген айнымалылар анықталған:

1. $? — соңғы команданы қайтару коды. Сценарийден қайтару exit командасымен қайтару кодын көрсету арқылы жүргізіледі:

```
(exit <қайтару коды>)
```

Қайтару коды тапсырманы орындауды басқару үшін қызмет етеді, мысалы файл ішінде жолақты тізбектей іздеген кезде керек. Бағдарламаның қайтару кодына байланысты grep файлда жолақты іздеуде егер қажет жолақ табылмаса қателік туралы хабарлама беруге болады немесе келесі келесі жолақты іздеуді жалғастыру керек.

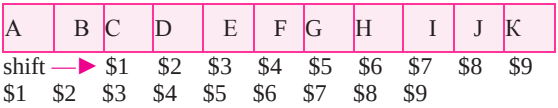
2. $# — тапсырманы шақыру командалық жолағының параметрлер саны;

$1, ..., $9 — бұл айнымалылардың көмегімен тапсырманы шақырған командалық жолақта параметрлерге мән беру орындалады. Айнымалы $1 бірінші параметрге сәйкес келеді, айнымалы $9 — тоғызыншы.

Тоғызыншыдан кейін егер параметрлерге қолжетімділікке қажеттілік туындаса, параметрлерді тізім бойынша тоғыз параметр бойынша «терезені» оң жаққа жылжытатын shift командасы қолданылады (сурет-5.1).

shift командасын орындағаннан кейін, екінші айнымалы $1айнымалы арқылы қолжетімді болады, ал онынші айнымалы—$9 командасы арқылы қолжетімді болады. Shift командасын орындау саны шексіз. Дегенмен командалық жолақтың соңғы параметрі айнымалы мәніне $1 ауыстырып қойған кезде алынады, ал $2, ..., $9 айнымалыларының мәндерін алмастырып қойған кезде тек бос жолақтар алынатын болады. Параметрлерді кері қайтарып жылжыту қарастырылмаған.

Көп жағдайда тапсырманы айнымалы атауынан және белгіленген бөліктен тұратын жолаққа жазу қажеттілігі туындайды, мысалы $ABC түрдегі жолақ, бұл жерде $A — айнымалы, ал BC —белгеленген мәтіндік жолақ. Осындай жазба түрінде BASH айнымалы атауының соңы мен мәтіндік жолақтың басын ажырата алмайды, өйткені A айнымалысы және BC жолағы немесе AB айнымалысы және C жолағы екені немесе тіпті ABC айнымалысы екені белгісіз.



Сурет-5.1. Параметрлер терезесінің жылжуы

Дәл осы мәселеге байланысты айнымалы жүйелерінің $1, ..., $9 көмегімен 9 командалық жолақтан көп параметрлерге жүгіну мүмкін емес; $19 жазбасынан командалық жолақтың 19-параметрі сұратыла ма немесе артында «9» мәтіндік жолақ тұрған бірінші параметр қажет болды ма, ол жағы белгісіз.

Осы мәселенің шешімін табу үшін BASH (2.0 және одан жоғары) жаңа нұсқаларында айнымалылар атауын ерекшелеу таңбалары қосылған —фигуралы жақша. Айнымалыны шақыру үшін синтаксис ${айнымалы аты} қолданылады.

Атауын ерекшелеу таңбасының көмегімен 9 көп параметрлерге қолжетімділік мүмкіндігі бар: ол үшін айнымалы атауының орнына параметр саны көрсетіледі (${n} түрінде, бұл жерде n — кез-келген толық сан). Сондай-ақ ${n} көмегімен параметрлерге қолжетімділік

кезінде shift командaның әрекеті ескеріледі:

- $* — бұл айнымалыда тапсырмаға берілген командалық жолақтың барлық параметрлері сақталады. Бұл айнымалыда барлық параметрлер бір жақшаға алынған, яғни $* = "$1 $2 $3
- $@ — бұл айнымалыда тапсырмаға берілген командалық жолақтың барлық параметрлері сақталады. Бұл айнымалыда әр берілген параметр жеке-жеке жақшаға алынған, яғни $@ = "$1" "$2" "$3" ...;
- $0 — бұл айнымалыда орындалып жатқан тапсырманың файл атауы сақталады. Оның көмегімен файл атауына тәуелді емес тапсырманы рекурсивті шақыруды ұйымдастыруға болады. Басқаша айтқанда, шақырту әрқашан сол тапсырманы оның атауына тәуелсіз рекурсивті шақырады.

```
exec $0
```

жүйелік айнымалыларды пайдалану мысалы ретінде BASH тілінде келесі тапсырманы келтіреміз:

```
#!/bin/bash

while [ "$1" != "" ] ; do
echo $@
shift
done
```

Айнымалыдағы тапсырма echo командaның көмегімен оған берілген барлық параметрлерді шығарады. Әр шығарған сайын параметрлер терезесі shift командасының көмегімен жылжиды. Тапсырманың орындалуы параметрлер тізімі таусылғанда аяқталады, яғни shift командaның соңғы орындағаннан кейін бірінші параметр бос жолаққа тең болып қалады (Қолданылған командалар туралы толығырақ 2-қосымшадан қараңыз).

Тапсырманы орындау нәтижесінде экранға келесі шығатын болады:

```
$ ./test.sh 1 2 3 4 5
1 2 3 4 5
2 3 4 5
3 4 5
4 5
5
```

Әр жаңа айналым итеративті жүйелік айнымалының $@ мәні параметрлер терезесінің ағымдағы орналасуына сай өзгеретінін және қолжетімді параметрлердің жалпы саны біртіндеп азаятынын көруге болады.

5.4.3. Тапсырма айнымалыларын ортаға көшіру

Тапсырманы орындау барысында «=» операциясы көмегімен мәндер меншіктелген айнымалылар тек тапсырма ішінде және тек тапсырманың орындалу кезінде қолжетімді. Осындай айнымалылар жергілікті тапсырманың орындалуында сыртқы ортадан оқшауланған болып қарастырыла алады.

Кейбір тапсырмалармен инициализацияланған айнымалыларды дәл сол командалық интерпретатор ортасында орындалатын басқа тапсырмаларға қолжетімді ету үшін айнымалыларды ағымдағы тапсырманың ортасына көшіру командасын қолдануға болады.

Тапсырма ортасына айнымалыларды көшіру export командасымен жүргізіледі. Оны шақыртудың екі түрі бар:

1) export <айнымалы аты> — инициализацияланған айнымалыны тапсырма орындалу ортасына көшіреді;

2) export < айнымалы аты > = <мән> — айнымалыға мән меншіктейді және айнымалылар ортасының алаңына көшіреді.

Тапсырманы орындау ортасында жарияланған айнымалылардың жиынтығы әдетте осы тапсырманың айнымалылар ортасының жинағы деп аталады. Барлық жарияланған айнымалыларды қарау үшін set командасы қызмет етеді:

```
$ set
PATH=/bin:/sbin:/usr/sbin
PWD=/home/nick
TTY=/dev/tty6
```

Командалық интерпретатордың жаңа көшірмесін жіберуде (мысалы, орындалатын bash файлын шақыру көмегі арқылы) анықталған барлық айнымалылар ортасы қалады. Сонымен қатар, командалық интерпретатор дәл сол айнымалылардың жаңа көшірмесін алады. Командалық интерпретатор ортасындағы айнымалы мәндерінің өзгеруі тек өзінің орындалу ортасына әсер етеді, бірақ шақырып отырған командалық интерпретатордың орындалу ортасын өзгертпейді.

Осылайша, пайдаланушымен келесі диалогта A мәнімен айнымалы ортасы анықталады, одан кейін командалық интерпретатордың жаңа көшірмесі іске қосылады, онда айнымалының мәні MYVAR иеленеді, содан соң қайта анықталады. Жаңа командалық интерпретаторды аяқтағаннан кейін және алдыңғы мәнге қайта келгесін MYVAR айнымалысы қайта қалпына келеді. Келесі мысалда пайдаланушымен

диалогтық режим жұмысы қарастырылатын болғандықтан жартылай қою әріппен жүйенің шығарған нәтижесі ерекшеленеді, жартылай қою курсивпен — пайдаланушының енгізулері белгіленеді, жақшадағы курсивпен орындалу барысына түсініктеме жазылады:

$ export MYVAR=A $ echo $MYVAR A

$ bash *(командалық интерпретатордың жаңа көшірмесін іске қосу)*

$ echo $MYVAR

A *(MYVAR мәнін иеленген)*

$ export $MYVAR=B $ echo $MYVAR

B *(MYVAR мәні ағымдағы интерпретатор ортасында қайта анықталған)*

$ exit *(алдыңғы интерпретаторға шығу)*

$ echo MYVAR

A *(мән қайта қалпына келген)*

Кейбір айнымалы қоршаулар пайдаланушы жүйеге кірген бойда іске қосылатын негізгі командалық интерпретаторды орындау ортасында жарияланады (8.2-тарауды қараңыз). Бұл айнымалы мәнін пайдаланушының жұмыс сеансы кезінде іске қосылған командалық интерпретатордың кез-келген көшірмесінде қолдануға мүмкіндік береді.

Мұндай айнымалы қоршауға PATH мысал бола алады. Бұл айнымалының мәні каталогтар атауының абсолютті тізімі болып табылады. Каталогтар атауының абсолютті тізімі егер файлдарды іске қосқан кезде тек файлдың атауы ғана беріліп, оған қатысты немесе абсолют жолды көрсетпеген болса орындалып жатқан файлдар (тапсырмалар мен бағдарламалар) іздестіру жүргізгенде қолданылады. Каталогтар тапсырмаларының тәртібі іздеу стратегиясын қалыптастырады, яғни командалық интерпретатор каталогтарын қарау тізбегі. Қарау тізімі егер дискте бір атаумен бірнеше орындалатын файлдар бар болған жағдайда қажет. Егер нақты каталог берілмесе алдымен орындалатын PATH айнымалысы тізімінің басына жақын орналасқан каталогтың файлы жіберіледі.

Пайдаланушыда өзінің орындалатын файлдары бар каталогы болуы мүмкін және ол каталогты тізімге қоса алады:

```
export PATH=$PATH:/check/scripts
```

Осылайша, PATH айнымалысына келтірілген мысалда «:» бөлгіш таңбасымен тіркескен оның ескі мәні меншіктеледі және каталогтың жаңа атауы мен орындалатын файлдар жазылады /check/scripts. Осы ретте PATH айнымалысының жаңа, қайта анықталған мәні пайдаланушының командалық интерпретаторы ортасында болады және пайдаланушының жүйемен жұмыс сеансы барысында ғана қолжетімді

болады. Басқа пайдаланушылар өз айнымалыларының мәндерін өздерінің анықталған командалық интерпретаторлары орталарында көретін болады.

Айнымалы қоршауларының мәндері пайдаланушы тапсырмасына ықпал етуі мүмкін болғандықтан, мысалы пайдаланушының файлдары орналасқан каталогтардың атауларын анықтау, олар пайдаланушы сеансының ақпараттық ортасына кіретін болады. Егер бұл ретте қандай да бір айнымалылар тапсырманың орындалуына әсер ететін болса олар сол тапсырманың ақпараттық қоршауына кіретін болады.

Мысалы, «Білімді басқару» жүйесінің құрамына кіретін тапсырмалардың көп бөлігі құрамында жүйенің файлары бар каталог аты ретінде BASEDIR айнымалының мәнін қолданады. Егер бұл айнымалы анықталмаған болса, онда тапсырма орындалуын өзі тоқтатады:

```
if [ "${BASEDIR:-DUMMY}" == "DUMMY" ] ; then
echo "Переменная \$BASEDIR не задана" exit
100 fi
```

5.4.4. Айнымалылардың мәндеріне қолжетімділік

Айнымалының мәнін меншіктеу келесі құрылым көмегімен жүзеге асады:

`<айнымалының аты>=<мән>` немесе

`let <айнымалының аты>=<мән>`

Мәнді меншіктеуде ескере кететін жағдай, айнымалы атауы мен теңдік белгісі арасында, сондай-ақ теңдік арасы мен мән арасында бос орын қалдырылмайды.

Тапсырманы орындау кезінде айнымалы мәніне қолжетімділік алу үшін «$» алмастырып қою операциясын қолдану қажет. Тапсырма мәтінінде айнымалының атауын тікелей көрсету жолақ ретінде қабылданады. Егер атауды алмастырып қою операциясымен алдын алса, тапсырманы орындау кезінде айнымалы мәнін алмастырып қою «$» таңбасынан кейін атауды көрсету арқылы жүргізіледі. Оны мысал ретінде түсіндірейік:

```
#!/bin/bash
variable=Hello
```

```
echo variable
echo $variable
```

Осындай тапсырманы орындап болған соң, экранға шығады:

```
variable
Hello
```

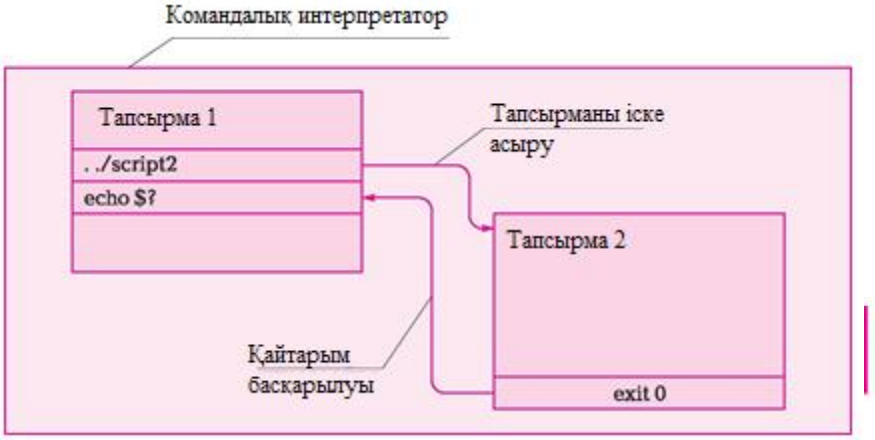
Жоғарыда айтылғандарды ескере келе бір айнымалының мәнін екінші айнымалының меншіктеп алуы төмендегідей жазылатынын білу қиын емес:

```
var1=$var2
```

5.5. ТАПСЫРМАНЫ ОРЫНДАУҒА ЖІБЕРУ

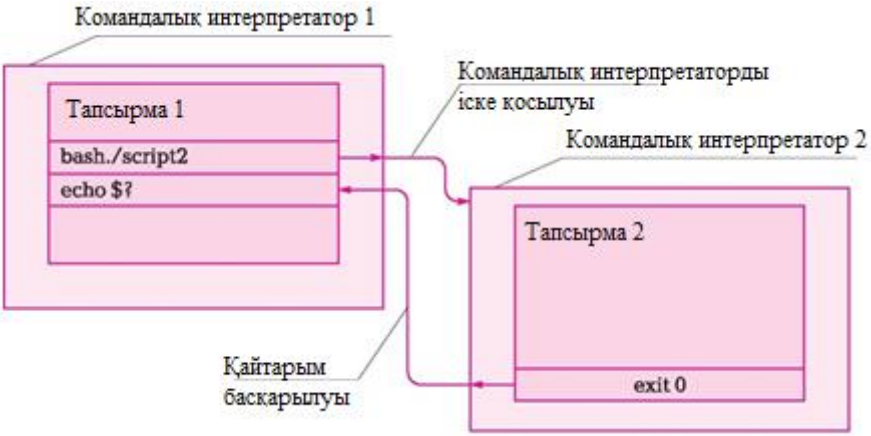
Тапсырманы орындауға жіберу үшін оны командалық интерпретаторға жіберу керек. Ол тапсырма мәтінін талдайды және тапсырма командаларын операциялық жүйенің жүйелік шақыртуына ауыстырады. Бұл ретте командалық интерпретатор тапсырманың ақпараттық қоршауын қолдайды, яғни тапсырманы орындауға қажетті барлық уақытша деректерді сақтайды, мысалы ағымдағы терминалмен байланысу, жергілікті айнымалылар, файлдық дескрипторлар. Тапсырма ағымдағы командалық интерпретаторға орындауға берілуі мүмкін — бұл кезде ағымдағы ақпараттық қоршау сақталады (сурет-5.2).

Сонымен қатар, ағымдағы командалық интерпретатор құралдарымен командалық интерпретордың жаңа процесін құрастыра



алады (сурет-5.3).

Сурет-5.2. Ағымдағы командалық интерпретатор мәтінінде тапсырманы іске қосу



5.3-сурет. Жаңа командалық интерпретаторда тапсырмалардың іске асырылуы

Бұл кезде басқару командалық интерпретатордың жаңа процесіне беріледі, ол өз кезегінде өзіне жүктеген тапсырманы орындайды. Тапсырманы орындап болған соң командалық интерпретатордың жаңа процесі басқаруды оны шақыртқан командалық интерпретаторға кері қайтарады. Жаңа командалық интерпретатордың ақпараттық ортасы да жаңа болады, бірақ ол осыған дейін болған командалық интерпретатордың ақпараттық қоршауының кейбір қасиеттерін өзіне қалдырады, мысалы, айнымалылардың мәндері, тапсырмамен өзгеретін файлдың ішіндегі ақпараттар және т.б.

Тікелей пайдаланушы тапсырманы орындауға командалық жолақтан жібере алады немесе басқа тапсырма арқылы жіберуге де болады. Екінші жағдайда іске қосылатын тапсырманың аты оны іске қосып жатқан тапсырманың мәтінінде команда ретінде жүреді. Іске қосатын тапсырманы аталық деп, іске қосылып жатқан тапсырманы еншілес тапсырма деп аламыз. Жаңа командалық интерпретатор жасау кезінде де солай, туындататын интерпретаторды жаңа пайда болған интерпретаторға қатысты аталық, жаңадан пайда болған интерпретаторды еншілес деп аламыз.

Тапсырманы жіберудің келесідей нұсқалары бар:

1) Тапсырма файлының атын көрсету арқылы (толық атауы немесе файлға қатысты жолды):

/check/scripts/teacher/gather.sh

Іске қосудың осы тәсілін пайдалану үшін тапсырмасы бар файлдың «орындалатын» атрибуты болуы қажет.

Тапсырманы жіберудің осы түрінде командалық интерпретатордың жаңа көшірмесі іске қосылады, оның жолы тапсырма тақырыбының атауында көрсетілген. Тапсырма аяқталған соң басқару аталық командалық интерпретаторға қайта оралады. Егер тапсырма басқа тапсырмадан жіберілген болса аталық тапсырма өз орындалуын келесі командамен жалғастырады. Егер тапсырманы пайдаланушы командалық жолақтан жіберген болса аталық интерпретатордың командалық жолағына шақыртудың қайтарылуы болады.

Егер осы тәсілмен іске қосылатын тапсырма жолағында тапсырма файлының жолы толық немесе қатысты көрсетілмесе, тапсырма файлын іздестіру әдеттегі орындалатын файлды іске қосатындай PATH айнымалылар қоршауында келтірілген каталогтарда жүргізіледі;

2) Тапсырма файлының атын параметр ретінде көрсету арқылы командалық интерпретаторды жіберу жолымен:

```
/bin/bash /check/scripts/teacher/gather.sh
```

Осы тәсілмен жіберген кезде тапсырмасы бар файлдың «орындалатын» атрибуты болуы шарт емес. Сонымен қатар тапсырма мәтінінде тақырыптың атауы болу қажеттілігі жоқ — командалық интерпретаторды таңдауды тапсырманы орындалуға жіберетін пайдаланушы жасайды, сол себепті орындауға жіберудің бұл тәсілінде тақырып ат ескерілмейді.

Орындауға жіберудің бірінші тәсіліне қарағанда, тапсырма файлына абсолют немесе қатысты жолдың болмауы жағдайында, оны іздестіру ағымдағы каталогта жасалады. Ол бұл жағдайда файл аты командада интерпретатордың параметрі ретінде беріліп, орындалып отырған файл аты болып пайдаланылмайтынына байланысты. Басқасынан жіберу тәсілі алдыңғы тәсілге ұқсас;

3) exec командаңы көмегімен жүзеге асыру:

```
exec /check/scripts/teacher/gather.sh
```

Осылай жіберілу кезінде басқару еншілес командалық интерпретаторға қайтарымсыз беріледі— іске қосылған тапсырманы орындайтын аталық командалық интерпретатор толық еншілеске ауысады. Еншілес тапсырманың орындалуы аяқталғаннан кейін аталық тапсырманы орындауға қайта ауыстырылмайды. Тапсырманың файлы іске қосудың осы түрінде «орындалатын» атрибутына ие болуы керек;

4) сол командалық интерпретаторды жүзеге асыру жолымен:

```
. /check/scripts/env.sh
```

Егер жүргізілетін тапсырма атауының алдына бос орын тастап, нүкте қойса, онда тапсырманың орындалуы іске қосылған командалық

интерпретатор көшірмесімен жалғасады, сонымен қоса жаңа тапсырмаға дәл сол орындалу ортасы пайдаланылады.

Осылай іске қосқан кезде жаңа тапсырма өзінің қолдануына оны шақырған тапсырма пайдаланудан айнымалыларды алатынын түсіну қиын емес — командалық интерпретатордың көшірмесін сақтаған кезде оның ішкі күйі де және айнымалылары да сақталады. Шақыртылған тапсырмалардағы жарияланған және инициализацияланған барлық айнымалылар, ол аяқталғаннан кейін де, шақырған тапсырмаға басқару берілген кезде де сол мәндерге ие болады.

5.6. ЕНГІЗУ/ШЫҒАРУ. КОНВЕЙЕРЛІК ӨҢДЕУ

Тапсырманы орындау кезінде терминалға шығатын және терминалдан алынатын деректер ол жерге тікелей түспейді. Деректерді алу және жіберу үшін енгізу/шығару буфері қолданылады — оперативті жадының аралық бөлігі, ол жерде ОЖ терминалға тікелей жібермес бұрын және терминалдан алғаннан кейін деректерді жинайды. Енгізу/шығару буферлерін басқару операциялық жүйенің ядросына кіретін енгізу/шығару ішкі жүйесінің пайдаланушысынан тәуелсіз жүреді. Енгізу/шығару буферлеріне қолжетімділік файлдарға қолжетімділікке ұқсас жүргізіледі: қалыптасқан ереже бойынша енгізу/шығару жүйесі үш виртуалды файлдарға ашық қолжетімділік жасайды, олардың екеуі деректерді экранға шығару үшін қызмет етеді, ал біреуі – терминалдан ақпарат алады. Бұл файлдар енгізу/шығару ағынының атауын алды:

- stdout — шығару ағыны, әдепкі қалпы бойынша — терминал экраны;
- stderr — қатені шығару ағыны, әдепкі қалпы бойынша — терминал экраны;
- stdin — енгізу ағыны, әдепкі қалпы бойынша — пернетақта.

Енгізу/шығару ағынына түсетін ақпараттар, әдетте терминал құрылғысына сәйкес келетін /dev каталогтағы файлға бағытталады. Тапсырманы орындау кезінде пайдаланушы деректерін файлға алып кету үшін немесе файлдан кіріс ақпараттарды алу үшін енгізу/шығару қайта бағытталуы ықтимал. Осы сәтте әдетте экранға шығарылатын ақпараттар көрсетілген файлға қайта бағытталады, ал әдетте пернетақта арқылы енгізілетін деректер файлдан алынады. Қайта бағыттау қосылып тұрған кезде экранға деректерді шығару немесе

оларды пернетақта көмегімен алу жүзеге асырылмайды, яғни файлдар толықтай терминал құрылғысын алмастырады. Деректерді қайта бағыттауды басқару енгізу/шығаруды қайта бағыттау команданың көмегімен жүргізіледі.

Бағдарлама шығарып жатқан деректерді (оның ішінде тапсырмалар да бар) file.txt файлына қайта бағыттау үшін шығаруды қайта бағыттау таңбасын «>» көрсетіп бағдарламаны жүктеу керек:

```
prog > file.txt
```

Егер file.txt файл жоқ болса, осындай қайта бағыттау кезінде ол жаңадан пайда болады да, терминалға шығуы керек барлық деректер соның ішіне жиналады. Егер file.txt файл бар болса, оның ішінде қайта бағытталуды жүктегенге дейін болған ақпараттар жаңа деректермен өшіріліп тасталады.

Бірнеше бағдарламалардың алатын деректерді бір файлға жинаған кезде деректерді файлда бұдан бұрын болған деректер астына орналастыру қажет болады. Ол үшін шығаруды қайта бағыттау таңбасын пайдалануға болады «>>»:

```
prog >> file.txt
```

Бағдарлама шығарған деректерді осылай шақырған кезде олар файл file.txt соңына қосылатын болады. Егер бағдарламаны шақырған кезде ондай файл жоқ болса, оған деректер енгізер алдында жаңасы пайда болады.

Енгізу ағынын қайта бағыттау үшін енгізуді қайта бағыттау белгісін қолдану қажет «<». Енгізуді file.txt файлынан қайта бағыттау үшін шақырту келесідей болады:

```
prog < file.txt
```

Егер бір мезетте infile.txt файлынан енгізу және outfile.txt файлынан шығару қайта бағыттаулары қажет болса, онда шақырту келесідей болады:

```
prog < infile.txt > outfile.txt
```

Бір бағдарламаның шығару ағынын екінші бағдарламаның енгізу ағынына қайта бағыттау үшін енгізу/шығару конвейерін қолдануға болады. Конвейерді бекіту «|» таңбасымен жасалады. Осылайша, prog1 бағдарламасын шығаруды prog2 бағдарламасының енгізуіне қайта бағыттау келесідей болады:

```
prog1 | prog2
```

Енгізу/шығару конвейері UNIX - жүйесінің негізіне салынған

декомпозиция қағидасының негізгі ережелерін пайдаланудың қарапайым мысалы. Кез-келген күрделі тапсырмалардың әрқайсысы өзінің спецификалық тапсырмасы орындалатын бірнеше тізбектей жасалатын деңгейлерге бөлінуі мүмкін. Бұндай тәсілді пайдалану өте тиімді, өйткені бір әрекет бірнеше тапсырмаларда қатар қолданыла алады. Мысалы, екі тапсырманы алайық— біріншісі, тапсырмалардың алынған нұсқасының тізімін қарап, оны алфавиттік қатармен шығарады, екіншісі берілген тақырып бойынша нұсқа тізімін қарап оны алфавиттік қатармен шығарады.

Тұтас бағдарлама ретінде орындалғандықтан бұл екі тапсырма бір-бірінің функционалдығын қайталайды. Егер біз тек екі әрекетті қолдансақ— мәтіндік файлды шығару ағынына шығару және кіретін ағымдағы шығатын деректер ретінде дайындап саралау. Біз бөлек командалардан декомпозицияланатын екі қарапайым тапсырманы ала аламыз.

Ол кезде бірінші тапсырма келесідей түрленеді:

```
ls /check/students/vasya | sort
```

екіншісі —

```
ls /check/teacher/themel | sort
```

Бұл жерде ls командасы берілген каталогта файлдар тізімін шығарады, ал sort команданы осы ағында іріктейді де, сұрыпталған нәтижені қалыпты жағдайда шығару ағынына жібереді.

5.7. АЛМАСТЫРЫП ҚОЮ

5.7.1. Бағдарламаның қорытындысын алмастырып қою

Тапсырманы жазған кезде қандай да бір бағдарлама енгізетін ақпаратты сақтап қою қажеттілігі туындайды. Кейін тапсырманың орындалу барысында осы деректерді салыстырмалы түрде оңай қолдану үшін оларды тапсырма айнымалысында сақтауға болады. Ол үшін $( ) және ' ' (кері жақша) қолдану керек. Тапсырма мәтінінің ішіне кіргізілгеннен кейін олар тапсырма орындалу кезінде команда шығаратын деректерге ауысады.

'команда' жазбасының пішінін BASH командалық интерпретаторының барлық нұсқалары қолдайды. Мысалы,

```
var='ls /check'
```

команда /check каталогта бар барлық файлдар тізбегіне сай var мәнінің айнымалысын иемденеді.

Жазбаның жаңа нұсқаларын $(команда) тек BASH 2.0 және одан жоғары нұсқалар сүйемелдейді және бір-біріне кіріктірілген алмастырып қоюларды жасауға ықпал етеді. Мысалы, команда

```
var=$(ls /$(ls /check))
```

команда /check каталогта бар барлық файлдар тізбегіне сай var мәнінің айнымалысын иемденеді.

5.7.2. Топтық таңбалар

Алдыңғы бөлімде қарастырылған каталогтағы барлық файлдардың тізімін алу тапсырмасы ауыр болуы мүмкін. Пайдаланушыға каталогтағы барлық файлдардың тізімі емес, тек белгілі бір критерий бойынша анықталатын файлдар қажет болуы мүмкін, мысалы атауы А әрпінен басталатын немесе 8 таңбадан аспайтын файлдар қажет болуы мүмкін. Осындай өлшемдерді анықтау үшін файл атауының маскасы немесе жай маска қолданылады. *Маска* дегеніміз файл атауына қойылатын секілді шектеулер жасалған мәтіндік жолақ. Масканың басты ерекшелігі оның құрамына фай атауында пайдалануға болмайтын алмастырып қою таңбалары кіре алады. Файл атауының маскаға сәйкестігін тексергенде алмастырып қою таңбаларын атаудың бір немесе бірнеше таңбасы өзгереді.

Көп қолданылатын алмастырып қою таңбалары «*» және «?».

«*» алмастырып қою таңбасы файл атауында оның орнында таңбаның кезх-келген саны тұра алады дегенді білдіреді. Осылайша, text*.doc маскасына text1.doc, text123.doc және тіптен text.doc файл атаулары қанағаттандыратын болады.

«?» алмастырып қою таңбасы, оның орнында файл атауында бір таңба болады немесе ешқандай таңба қойылмайды дегенді білдіреді. Осылайша, text?.doc маскасына text1.doc, text.doc файл атаулары қанағаттандырылады, бірақ text12.doc атауы қанағаттандырылмайды.

Масканы қанағаттандыратын файлдар тізімін алу үшін маска ls команданың параметрі ретінде көрсетіле алады. Мысалы, ls *~ команда ағымдағы каталогтан атауының соңы мәттегі (әдетте ішіндегі деректері ескірген файлдар осылай аталады) аяқталатын барлық файлдарды экранға шығарады.

ТАПСЫРМАНЫҢ ОРЫНДАЛУ ҚАДАМЫН БАСҚАРУ

5.8.1. Командалардың орындалу тізбегі

Ең қарапайым тапсырмалар пайдаланушының командаларын тізбекпен орындау, сонымен қатар алдыңғы командалардың орындалу нәтижелері келесі командалардың орындалуларына еш әсер етпейді. Сонымен қатар, командалар қалай орындалып жатқандығы ескерілмейді. Жоғарыда қарастырылған қарапайым тапсырмаларда командаларды орындау кезегімен жүргізіледі.

Командалардың қалай орындалып жатқандығын анықтау қажет болса — тізбектей және параллель — бөлгіш-таңбаларды пайдалану қажет «;» және «&».

5.8.2. Командалардың параллель орындалуы

Бірнеше командаларды тізбек бойынша орындау үшін олар «;» таңбасымен ажыратылады. Нәтижесінде кезекті команданы орындау тек алдыңғы команданың жұмысы аяқталғаннан кейін ғана басталады. Егер «;» бөлгішінің орнына командалар арасына «&» бөлгіш-таңба қолданылса, онда бөлгішке дейін тұрған әр команда фондық режимде орындалады, ал келесі команда алдыңғы команда іске қосылған сәтте бірден орындалады. Осылайша, параллель іске қосу және екі немесе одан да көп командаларды қатар орындау мүмкіндігі бар.

Екі командалардың тізбектей орындалуына арналған өрнек синтаксисі келесідей анықталады:

команда1 ; команда2

Параллель орындалу үшін де солай жасаймыз:

команда1 & команда2

Бөлгіш-таңба «;» әдетте тек бірнеше команданы бір жолаққа орналастыру қажеттілігі туындағанда қолданылатынын ескере кеткен жөн. Егер бөлгішті «;» жолақ соңындағы таңбаға өзгертсек те сол нәтижені алуға болады.

Егер «&» бөлгішін қолданған кезде, тек бірінші командасына көрсетсе (команда &), онда команда фондық режимде орындалады, ал орындалуы келесі жолақта орналасқан тапсырма командасына беріледі. Егер фондық режимде команданы іске қосу командалық жолақтан

жүргізілсе, онда команда өз орындалуын фондық режимде бастайды, ал басқару бірден командалық интерпретаторға беріледі.

5.8.3. Командалардың шартты орындалуы

Әр команданың орындалуына жіберу алдыңғы команданың орындалу нәтижесінен (қайтару коды) тәуелді болатын командаларды кезегімен орындау үшін && және II бөлгіші қолданылады.

Команда1 орындау үшін және егер ол сәтті орындалған болса, команда2 орындау үшін келесі жазба қолданылады:

```
команда1 && команда2
```

Команда1 орындау үшін және егер ол сәтсіз орындалған болса, команда 2 орындау үшін келесідей ұқсас жазба қолданылады:

```
команда1 | | команда2
```

Команданың сәтті орындалуы қайтарылу кодына байланысты анықталады. Сонымен қоса, нольге тең қайтару коды команданы сәтті орындады деген мағынаны береді; нольге тең емес қайтару коды – сәтсіз.

Ішінде && және || бөлгіштері бар командалық жолақ логикалық өрнек ретінде қарастырылады, оның мәні командалық интерпретатормен команда орындалуы бойынша есептеліп шығарылады. && және || бөлгіштер сәйкесінше логикалық қосу және логикалық көбейту операциясы ретінде қарастырылады. Логикалық атқарымдар аргументі ретінде команданы қайтару кодтары пайдаланылады. Қайтарудың нольдік коды нақты мәнге сәйкес келеді, нольден ерекше код— жалған мән.

Бөлгіштердің бұндай интерпретациясында олардың жұмыс жасау ережелері логика тілімен сипатталуы— егер && операциясымен бөлінген алғашқы екі команданың жұмысының қайтару коды нольге тең емес болса (жалған мән), демек логикалық атқарымның барлық мәндері жалған болады және екінші команданы орындау қажеттілігі жоқ. Сәйкесінше, егер || операциясымен бөлінген алғашқы екі команданың жұмысының қайтару коды нөлге аяқталса (нақты мән), демек логикалық атқарымның барлық мәндері нақты болады және екінші команданы орындау қажеттілігі жоқ.

|| және && операцияларының интерпретациясы солдан оңға қарай жүргізіледі, сонымен қоса олардың артықшылықтары бірдей. Артықшылықты өзгерту үшін дөңгелек жақшаларды пайдалану талап етіледі. Осылайша, command1 || command2 && command3 өрнектері (command1 || command2) && command3 өрнектеріне тең, бірақ

command1 || (command2 && command3) өрнектерімен тең емес.

```
команда! > файл; команда2 >> файл
```

5.8.4. Бағдарламаның шығару ағынын қосу

Командаларды шартты орындау тұрғысынан қарағанда фигуралық жақшалар топтастырылған таңбалары болып табылады. Дегенмен, олар бірнеше бағдарламалардың шығу ағынын біреуге қосу үшін қолданыла алады. Енгізу/шығаруды бірнеше командаға қайта бағыттау үшін жазуға болады.

```
{ команда1; команда2; } > файл
```

Сонымен қатар, жақшаға алынған командалардың тізбегін командалық интерпретатор деректері шығару ағынына түскен бір команда ретінде қарастырады.

5.8.5 Тапсырма айнымалыларының көріну аймағы

Дөңгелек жақшаларды ( ) пайдалану аймағы жай ғана командалар топтастырылуынан әлдеқайда ауқымды. Егер командалар тізбегін дөңгелек жақшаға орналастырсақ, командалар тізбегін орындап болған соң командалар өзгерткен айнымалылардың мәндері қайта қалпына келеді.

Мысалы, келесі тапсырма:

```
var="global"; (var="local"; echo "var is $var"); \ echo
"var is $var"
```

шығарады:

```
var is local var is global
```

Бұл жерде var айнымалысына алдымен "global" мәні меншіктеледі, одан кейін ол "local" өзгереді var айнымалысының мәнін өзгерткен команда дөңгелек жақша ішінде орналасқандықтан, ол орындалып болған соң бастапқы мәні қайта орнына келеді. Осылайша, дөңгелек жақша арқылы көріну аймақтарын басқаруға — тек тапсырманың жергілікті айнымалыларын және қоршаудың ғаламдық айнымалыларын да ажыратып қана емес, сонымен қатар бір тапсырма ішіндегі көріну аймағын анықтау.

5.8.6 Шартты оператор және айналым операторы

Тапсырманы орындау қадамын басқару тәсілдерінің ішінен кейбір тапсырмаларды жазу жеткіліксіз, мысалы, берілген өлшем бойынша файлдарды келесі айналымды өңдеу немесе ұйымдастыру үшін таңдау. Тапсырманы басқару тілінде объекттерді таңдау үшін шартты өрнек түсінігі қолданылады. *Шартты өрнек* — эталонды анықтайтын өрнек, оған қарап эталонға сәйкес келетін объектіні таңдау және эталонға объектінің сәйкес келу деңгейі анықталады.

Мысалы, K > 2 өрнегінде K айнымалысы сәйкестік анықталатын объекті болады, теңестіру операциясы «>» эталонға сәйкестік дәрежесін береді, ал тұрақты 2 — эталонның өзі. Шартты өрнектің тексеру нәтижесі егер объект эталонға берілген дәлдікпен сәйкес келсе логикалық шындық болады немесе логикалық жалған егер объект эталонға сәйкес келмесе.

BASH тілінде шартты өрнектерді тексеру үшін test команданы қолданылады. Оны шақыртудың екі форматы бар:

```
test <өрнек>
```

немесе

```
[ <өрнек> ]
```

Екі нұсқада да команда параметр ретінде көрсетілген логикалық өрнектің мәнін есептейді. Егер өрнек шын болса қайтару коды 0 қайтарады және егер жалған болса – 1 қайтарады. Өрнек арасындағы бос орынға және екінші мысалдағы тік жақшаға назар аударған— олар міндетті түрде орындалу керек, бос орынды тастап кету көптеген қателіктер туындатады.

Қасиеттері өрнектерде тексерілетін объектілері файлдар, жолақтар және сандар болуы мүмкін. Файлдар, жолақтар және сандардың қасиеттерін тексеру үшін кейбір өрнектердің форматтары төменде келтірілген. Ары қарай мәтінде материалды мазмұндау барысында test команданың көмегімен орындалатын қосымша тексеріс түрлері келтірілген.

- -z <жолақ> — жолақтың ұзындығы нольге тең (жолақ бос);
- -n < жолақ> — жолақтың ұзындығы 0 үлкен (жолақ бос емес);
- " жолақ1" == " жолақ2" — екі жолақ бір біріне тең;
- " жолақ1" != " жолақ2" — екі жолақ бір біріне тең емес;
- сан1 -eq сан2 — сандар тең;
- сан1 -ne сан2 — сандар тең емес;

- сан1 -lt сан2 — 1 саны 2 санынан кіші;
- сан1 -le сан2 — 1 саны 2 санынан кіші немесе тең;
- сан1 -gt сан2 — 1 саны 2 санынан үлкен;
- сан1 -ge сан2 — 1 саны 2 санынан үлкен немесе тең;
- -s <файл> — файлдың көлемі 0 көп(файл бос емес);
- -f <файл> — файл бар және ол қарапайым файл болып табылады;
- -d <файл> — файл бар және ол каталог.

Кез-келген өрнек алдында логикалық терістеу таңбасын қоюға болады «!»:

- ! <өрнек> — <өрнек> шын болған кезде, барлық өрнек жалған. өрнек>. <өрнек> жалған болған кезде, барлық өрнек шын.

Өрнектер логикалық ЖӘНЕ және логикалық НЕМЕСЕ операцияларының көмегімен біріге алады:

- < өрнек1> -a < өрнек2> — барлық өрнек шын, < өрнек1> ЖӘНЕ < өрнек2> өрнек шын болған кезде;
- < өрнек1> -o < өрнек2> — барлық өрнек шын, < өрнек1> НЕМЕСЕ < өрнек2> өрнек шын болған кезде.

Жоғарыда келтірілген өрнектер тармақталу командаларын тексерген кезде немесе тапсырма кезінде шарт айналымға негізделсе қолдануға болады. Мысалы, тармақталу блогының синтаксисі if шарты бойынша келесідей анықталған:

```
if <логикалық өрнек -1> ; then <командалар-1>
elif <логикалық өрнек-2> ; then
<командалар-2>
else
<командалар-3>
fi
```

Бұл жерде <команда-1>блогы <логикалық өрнек-1> шын мәнінде орындалады, <команда-2> командалар блогы <логикалық өрнек-2> шын мәнінде орындалады. Сонымен қатар бағдарламалаудың құрылымдық тілі Else If конструкциясына сәйкес elif өрнегінің көмегімен көптеген тексерістерге рұқсат беріледі. Барлық тексерілген логикалық өрнектер теріс болса, else негізгі сөзінен кейін <команда-3> блогы орындалады. Блок fi басты сөзімен аяқталады.

«;» таңбасы осы фрагментте командаларды бөлу таңбасы болып қолданылады, өйткені синтаксис тұрғысынан BASH if және then — бұлар әр түрлі командалар, оларды «;» таңбасымен бөлу керек немесе тапсырма файлында әртүрлі жолақтарға орналастыру керек.

Тармақталу команданы үшін логикалық өрнек ретінде кез-келген

команда орындала алады, ал мәні оның қайтару кодымен тексеріледі. Көбінесе бұндай команда ретінде жоғарыда қарастырылған test команда қолданылады.

Мысалы, тапсырманың келесі фрагменті тапсырмада берілген командалық жолақтың бірінші параметрі бос па соны тексереді. Егер, бірінші параметр бос жолақ болса, тапсырмаға командалық жолақтың бірде бір параметрі берілген жоқ деген хабарлама келеді және тапсырманың орындалуы қайтару коды аяқталады. Мұндай тексерістің ең оңтайлы түрі берілген параметрлер $# -eq 0 санын тексеру болып табылады. Тапсырмада параметрлерді беру позициялық болғандықтан, бірінші параметрдің бостығын тексеру рұқсат. Дегенмен мұндай тексеріс алғашқы параметр ретінде бос жолақты көрсету кезінде де жүзеге асады (экрандайтын таңбалар көмегімен – екі тырнақша):

```
test.sh "" A B
```

Екінші тексеріс берілген параметрлер санымен жұмыс жасайды — егер олар үштен көп болса, сәйкес хабарлама шығады және тапсырманың орындалуы 2 қайтару кодымен аяқталады:

```
if [ -z $1 ] ; then
echo "No command line parameters are specified"
exit 1
elif [ $# -gt 3 ]; then
echo "Too many parameters are specified"
exit 2 fi
```

Логикалық шарт циклды шектеуші ретінде де қолданыла алады. Осылайша while ... do ... done конструкциясында, төменде келтірілген синтаксис, логикалық өрнек шын болған кезде (0 тең) операторлар блогы орындалады <операторлар>:

```
while <логикалық өрнек> ; do
<оператор>
done
```

Кодтың келесі фрагменті тапсырмаға берілген барлық параметрлерді шығарады. Оған қоса жоғарыда айтылып өткен shift командаңы қарастырылады. Циклдың орындалуы бірінші параметрдің мәні бол болмағанша жалғаса береді. Сонымен бірге, айналымның әр итерациясында параметрлер терезесінің ығысуы және оларды қайта номерлеу болады (5.1-бөлімін қараңыз):

```
while [ ! -z $1 ] ; do
echo $1
```

```
shift
done
```

Берілген тізім мәндері бойынша циклды ұйымдастыру үшін for ... in ... do ... done конструкциясы қолданылады, жалпы синтаксисі төменде келтірілген:

```
for <цикл> in <тізім> do
<оператор>
done
```

Мәндер тізімі <тізім> бөлгіштері бар мәтіндік жолақ. Бөлгіштер ретінде бос орындар, табуляция таңбалары және жолақты ауыстыру таңбалары қолданылады. Циклде көрсетілген айнымалы тізімдегі элементтердің мәндерін тізбек бойынша иеленеді және операторлар <операторлар> блогында іске қосылуы ықтимал. Бұл цикл итерациясының саны тізім элементтерінің санына тең.

Тапсырманың келесі фрагменті алдыңғы фрагмент орындағандай әрекеттер жасайды, тапсырмаға берілген барлық параметрлерді шығарады. Тізім ретінде бұл жерде BASH кіріктірілген айнымалысы қолданылады, оның құрамында тапсырманың командалық жолағындағы барлық параметрлер бөлгіштер арқылы көрсетілген (4.2-бөлімді қараңыз):

```
for i in $@ do
echo $i
done
```

for циклының тізімі ретінде белгіленген мәндер өте сирек қолданылады. Әдетте тізім тапсырманы орындау кезінде мәнді алмастырып қоюмен генерацияланады. Жаңа ғана келтірілген мысалда айнымалы мәнін алмастырып қою қолданылған. Дәл осылай қандай а бір команданың шығарылуын алмастырып қою арқылы қолдануға болады. Мысалы, кодтың келесі фрагменті ағымдағы каталогтағы барлық файлдар ішіндегісін шығарады, әр шығару алдына өздерінің атауларын тіркейді:

```
for i in 'ls .' do echo === $i
=== cat $i done
```

5.9. WINDOWS ОПЕРАЦИЯЛЫҚ ЖҮЙЕЛЕРІНДЕ ТАПСЫРМАНЫ БАСҚАРУ ТІЛІ

5.9.1. Windows командалық интерпретатор

Дәл Linux секілді, Windows операциялық жүйелері командалық интерпретатор интерфейсімен жұмыс жасайды. Көбіне оның қасиеттері MS DOS операциялық жүйесінің командалық интерпретаторынан алынған және UNIX/ Linux операциялық жүйелеріне тән ерекшеліктермен толықтырылған. Бұл бөлімде Windows операциялық жүйелеріндегі Windows XP жүйесінде орындалған командалық интерпретаторға негізделген қосалқы жүйенің кейбір қасиеттерін қарастырамыз.

Windows операциялық жүйелерінде Windows NT ядросына негізделген командалық интерпретаторды іске қосу үшін cmd.exe бағдарламасы қолданылады. Командалық жолақ интерфейсін жүргізу үшін Іске қосу (Start) мәзірін ашу керек, мәзірден Орындау (Run) тармағын таңдап, бағдарламаны cmd.exe іске қосамыз.

Командалық интерпретаторды іске қосқан соң, стандартты жұмысқа шақыру терезесі ашылады. Windows XP операциялық жүйелерінде мұндай шақырту келесідей болады:

```
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\>
```

Бірінші екі жолақ пайдаланушы операциялық жүйенің нұсқасы туралы ақпараттандырады және Microsoft фирмасының артықшылыққа ие құқықтарымен таныстырады. Соңғы жолақ командаларды енгізуге арналған стандартты шақырту немесе пайдаланушыдан тапсырма алуға арналған шақырту болады.

Бұл интерпретаторда командаларды енгізу Linux операциялық жүйелеріндегі деректерді енгізумен бірдей. Мысалы, операциялық жүйенің нұсқасын келесі жолмен шығаруға болады:

```
C:\>ver <Enter>
Microsoft Windows XP [Version 5.1.2600]
C:\>
```

5.9.2. Windows пакеттік өңдеу

Тапсырмамен жұмыс жасау механизмдері мен құралдары Linux операциялық жүйелеріне қолданылатын тәсілмен ұқсас. Сонымен қатар тапсырма мәтіндік файлдар түрінде де рәсімделеді, ішінде командалық интерпретатормен орындалуы қажет командалар тізімі болады. Файлдардың өздерінде .bat немесе .cmd кеңейтілуі болуы шарт. Ал Windows орындалуы, параметрлерді беру механизмі және командалық сценарий жұмыстарының нәтижесі дәл Linux бірдей. Windows командаларды шақырту екі бөліктен тұрады:

```
<команда аты> <параметрлер>
```

Команда аты ретінде командалық интерпретатордың ішкі команданы қолданылады немесе сыртқы бағдарлама коды бар орындалатын файл аты алынады.

Параметрлердің максималды рұқсат етілген саны UNIX- ұқсас жүйелеріндегідей жолақтың максимал ұзындығымен анықталады.

Windows 2000 және Windows NT 4.0 операциялық жүйелері үшін жолақтың рұқсат етілген максимал ұзындығы 2 047 таңба, Windows XP және соңғы шыққан операциялық жүйелердің нұсқалары үшін 8 191 таңбаны құрайды.

Командалық интерпретатор қолдайтын командалардың толық тізімін алу үшін help команданы қолданылады. Қандай да бір команданың спецификалық қасиеттері туралы ақпарат алу үшін келесі команданы орындау қажет:

```
help <команда>
```

командалық жолақпен жұмысты қолдайтын барлық кіріктірілген командалар және көптеген сыртқы бағдарламалардың өзара кіріктірілген параметрлері бар. Осы параметрмен команданы шақыртқан кезде, басқа командаға беруге болатын оның арналуы және параметрлері туралы ақпарат шығады.

5.9.3. Айнымалар

Командалық сценарий жасаған кезде пайдаланушы айнымалы қоршаулармен операциялар орындауға мүмкіндігі бар. Айнымалылар бірнеше жерден келіп түсуі мүмкін. Айнымалы ортасының қай көзінен келіп түскенін анықтауға арналған айнымалылардың үш түрі бар: кіріктірілген жүйелік, кіріктірілген пайдаланушыға арналған және жергілікті.

Кіріктірілген жүйелік айнымалылар операциялық жүйе деңгейінде анықталады және пайдаланушыдан тәуелсіз барлық процестерге қолжетімді. *Кіріктірілген пайдаланушыға арналған айнымалылар* пайдаланушы жүйеге кірген уақытынан бастап анықталады және пайдаланушының жұмыс сеансы аяқталғанша болады. Ортаның барлық айнымалылар тізімін және олардың ағымдағы мәндерін алу үшін (жүйенің және пайдаланушының) set команданы қолданылады.

Кіріктірілген айнымалыларға қарама-қайшы *жергілікті айнымалылар* командалық сценарийді орындай деңгейінде анықталады. Жаңа айнымалыны анықтау үшін және оның ағымдағы мәнін өзгерту үшін тағы да set команданы орындалады:

```
set <айнымалының аты>=<мән>
```

Айнымалының ағымдағы мәнін алу үшін тек айнымалы атын көрсеткен жеткіліксіз. Сонымен қатар, жүйеге айнымалының нақ мәні керек екендігін көрсету қажет. Ол үшін «%» таңбалары айнымалы атының басында және соңында қолданылады. Бұл таңбаларды қолданбасақ жүйе айнымалы атын қарапайым жолақ ретінде қабылдайды.

Айнымалылармен жұмыстың бұл ерекшелігін келесі мысалда көрсетуге болады:

example1.bat
```
@set
variable=value @echo
variable @echo %variable%
```

Бұл мысалда echo команданы таңбалар тізбегі консольға шығару үшін қолданылады. Әр команда алдында болатын @ таңбасы командалық интерпретаторға команданы оның орындалуы алдында консольға шығарудың қажеттілігі жоқ екендігін хабарлайды. Егер бұл таңбаны көрсетпесе сценарийдің орындалуы кезінде консольға әр команда орындалар алдынды шығып отырады. example.bat файлы орындалып болған соң экранға төмендегідей жолақтар шығады:

```
C:\>example.bat
variable
value
C:\>
```

Осы мысалдан көрініп тұрғандай, бірінші жағдайда таңбалардың тізбегі variable жүйемен қарапайым таңбалық жолақ ретінде түсіндіріледі, екінші жағдайда сәйкес айнымалымен ұқсастырылған мән қайтарылады. Осылайша, тек қана жергілікті айнымалылардың

мәндеріне ғана емес кіріктірілген айнымалылардың мәндеріне де қолжетімділік алуға болады.

Айнымалыларға белгілі мәндерді меншіктеген кезде барлық таңбаларды тікелей қолдануға болмайды, өйткені кейбір таңбалар қатары резервтелген және жүйе оларды командалар немесе қызметтік символдар деп белгілейді. Оларға @, <, >, &, | жатады. Егер осы таңбаларды айнымалыларды белгілегенде пайдалану қажет болса, олар « » жақшаға алынулары тиіс.

```
set var=login^@e-mail.com
```

Айнымалылардың мәні ретінде тек жолақтарды емес, сандарды да, арифметикалық өрнектерді де қолдануға болады. Сандық мәндерді меншіктеу үшін set/a конструкциясы қолданылады. Математикалық өрнектерде тек -231 бастап 231 – 1 дейін диапазоны аралығындағы толық сандар қолданыла алады.

Өрнектерде арифметикалық операцияларды да қолдануға болады, олар + (қосу), - (алу), * (көбейту), / (бөлу), % (бөлгеннен қалған қалдық). Сонымен қатар біріктірілген, меншіктеу операторлары бар += (қосы және меншіктеу), - = (алу және меншіктеу), *= (көбейту және меншіктеу), / = (бөлу және меншіктеу) және % = (бөлуден қалдықты алу және меншіктеу).

```
set /a var=1
set /a res=%a%+%b%
set /a total+=1
set /a count*=(%amount%+1)
```

Алдында анықталған жергілікті айнымалыны өшіру үшін келесі команда қолданылады:

```
set <айнымалының аты>=
```

жергілікті айнымалылар командалық қабықтың тек ағымдағы данасына ғана және осы ағымдағы данадан туындаған қабықтың даналарына да қолжетімді. Бірақ егер де кейбір айнымалыларды қабықтың ағымдағы данасының деңгейінде ғана емес, басқа да жергілікті деңгейде локальды ету қажет болса, онда мына командаларды қолдануға болады: setlocal және endlocal. Айнымалы қабатында аумақтың ішінен пайдаланушы жасаған кез-келген өзгертулер, endlocal команданы орындалғаннан кейін іске аспай қалады. Осы командаларды пайдалану мысалын қарастырайық:

example2.bat

```
@set variable=global value
```

```
@echo Before setlocal @echo
%variable%
@setlocal
@set variable=local value
@echo After setlocal @echo
%variable%
@endlocal
@echo After endlocal @echo
%variable%
```

Осы сценарий жұмысының нәтижесінде консольға келесідей хабарламалар шығарылатын болады:

```
Before setlocal
global value
After setlocal
local value
After endlocal
global value
```

Жергілікті блоктан шыққаннан кейін пайдаланушы жасаған барлық өзгерістер еленбей қалды, ал айнымалылардың мәндері қайта қалпына келді.

Дәл Linux секілді пайдаланушы тек кіріктірілген және жергілікті айнымалыларға ғана емес, арнайы жүйелік айнымалыларға да қолжетімді. Бұл айнымалылар шақырылатын сценарийде берілетін параметрлерге қолжетімділікті орындауға мүмкіндік береді.

Windows командалық интерпретаторда арнайы айнымалылар %0..%9 анықталған. Айнымалы %0 сценарийдің орындалудағы атымен ығыстырылады, ал %1..%9 сценарий параметрлерінің алғашқы тоғыз параметрлермен ығысады.

Тоғызыншыдан кейінгі параметрге қолжетімділікке ие болу үшін shift командаңы қолданылады. Оның қызметі BASH ішіндегі аттас командаңың тәртібіне ұқсаған: бұл командаңы бір рет шақырғаннан кейін айнымалы %1 екінші параметрмен салыстырылады, ал %2 — үшіншімен және т.б.

Барлық параметрлердің тізімін алу үшін арнайы кіріктірілген айнымалы %* қолданылады. shift командаңы позициялық айнымалылардың %1, %2 және т.б. мәндеріне ғана ықпал етіп қоймайды, ол %* айнымалымен қайтып келетін мәнге де ықпал ететінін ескеру қажет.

Командалық жолақ параметрлері тікелей қолданумен қатар, оларға арнайы модификаторлар қолдануға болады. Бұл модификаторлар командалық жолақ параметрлерінен файлдар, каталогтар аттарын, сәйкес файлдар және т.б. жасау уақытын ерекшелеу үшін қызмет етеді. Модификаторларды қолдану үшін %~ тізбегі пайдаланылады, одан кейін осы модификатор қолданылатын параметрдің модификаторы мен позициялық номері жүреді. Командалық жолақ параметрлерінің модификаторлар тізімі 5.1-кестеде келтірілген.

Модификаторлар тек жеке емес, бір-бірімен бірлесіп те пайдаланыла алады. Мысалы, %~nx1 комбинациясы бірінші параметрді бөлшектейді, оның ішінен файл атын, кеңейтілуін алады, ал %~ftza1 комбинациясы консолға бірінші параметрді dir команданы шығаратын үлгіде шығарады.

Кесте-5 Командалық жолақ параметрлерінің модификаторлары

Модификатор	Сипаттама
%~	Параметр мәнін қайтарады. Егер қоршаған екі қайтара жақшалар бар болса оларды өшіреді.
%~f	Файлға параметр берген апарар жолды қайтарады
%~d	Файлға параметр берген сақталатын диск әрпін қайтарады
%~p	Файлға параметр берген апаратын толық жолды қайтарады
%~n	Файлға параметр берген атауын қайтарады
%~x	Файлға параметр берген кеңейтілуін қайтарады
%~s	Файлға параметр берген апарар жолды қайтарады, 8.3-форматын қолданады
%~a	Файлға параметр берген атрибутты қайтарады
%~t	Файлға параметр берген мерзімді және уақытты қайтарады
%~z	Файлға параметр тағайындаған өлшемін қайтарады
%~$PATH:	PATH айнымалысында анықталған каталогтар тізімін сканерлейді, және осы каталогтарда параметр тағайындаған файлдың атын іздейді. Егер ондай файл табылса, ең бірінші табылған файлға толық жол қайтарылады. Егер ондай файл табылмаса бос жолақ шығады

5.9.4. Енгізу/шығару. Конвейерлік өңдеу

Ақпаратты енгізу/шығару механизмін ұйымдастыру Windows консольдік қосымшаларында осы механизмді Linux ұйымдастыру механизммен ұқсас. Консольдан оқу немесе консолға жазу операциялары тікелей емес, құрылғылардың виртуал файлдары арқылы орындалады.

Нақ Linux секілді, бұл файлдардың стандартты атаулары бар және әрқайсысының артында стандартты файлдық дескрипторлар бекітілген, олар кез-келген консольдік қосымшалар орындалған кезде автоматты түрде ашылады. Бұл виртуалды файлдар енгізудің стандартты ағынын (0 дескрипторымен байланысты), шығарудың стандартты ағынын (1 дескрипторымен байланысты) және қателерді шығарудың стандартты ағынын (2 дескрипторымен байланысты) қамтиды.

Стандартты ағынмен қауымдастырылған енгізу/шығару файлдарына қайта бағыттау мүмкіндігі бар.

Бұл үшін енгізу/шығару қайта бағыттау операциялары қолданылады. Осы операциялардың синтаксисі мен семантикасы Linux пайдаланатынмен бірдей. Сол себепті оларды түсіндіру толығырақ тоқталмаймыз, бұл операцияларды бір 5.2-кестесіне біріктіреміз.

Сондай-ақ Linux бірдей бұл операцияларды біріктіріп командада бір уақытта бірнеше операцияларды қолдануға болады.

Кесте-5.2. енгізу/шығаруды қайта бағыттау операциясы

Операция	Сипаттама
>	*Команда1 > файл* Команданы стандартты шығару ағыны файлмен немесе құрылғымен байланысады. Егер ол файл бұрын жоқ болса, онда жаңадан құрылады. Егер бар болса, файл қайта жазылады
>>	*команда1 >> файл* Команданы стандартты шығару ағыны файлмен немесе құрылғымен байланысады. Егер ол файл бұрын жоқ болса, онда жаңадан құрылады. Егер бар болса, файл толықтырылып жазылады
<	*команда1 < файл* Команданы стандартты енгізу ағыны файлмен байланысады.
n >	*команда1 n> файл* файлдық дескрипторы бар команданы шығару ағыны n файлмен байланысады. Файлдық дескрипторлар 0 — 2 енгізу/шығару стандартты құрылғысымен байланысқан. Егер ол файл бұрын жоқ болса, онда жаңадан құрылады. Егер бар болса, файл қайта жазылады
n >>	*команда1 n>> файл* файлдық дескрипторы бар команданы шығару ағыны n файлмен байланысады. Егер ол файл бұрын жоқ болса, онда жаңадан құрылады. Егер бар болса, файл толықтырылып жазылады
n >& m	*команда1 n>& m* n файлдық дескриптор команданың шығыс ағыны m файлдық дескриптор ағынымен байланысады
n <& m	*команда1 n<&m* n файлдық дескриптор команданың кіріс ағыны m файлдық дескриптор ағынымен байланысады
|	*команда1| команда2* Бір процестің шығатын стандартты ағынын екінші процестің стандартты кірісімен байланыстырады

Мысалы, кейбір командалар шығаратын барлық деректерді стандартты шығару құрылғысы секілді және қатені шығарудың стандартты құрылғысы секілді қайта бағыттау үшін келесі команданы қолдануға болады:

```
команда >> output.log 2>&1
```

Осындай жолмен процессор аралық байланысты ұйымдастырудың бірнеше операцияларын біріктіруге болады:

```
команда1 | команда2 >log.txt
```

Бұл жағдайда команда1 стандартты шығысынан стандарт кіріс команда 2 деректер келіп түседі, ол стандартты команда 2 шыққан мәліметтер файл log.txt келіп жиналады.

5.9.5. Тапсырманы орындау қадамын басқару

Windows бағдарламаны орындау қадамын басқарудың бірнеше сан алуан механизмдерін қолдайды. Бұл командаларды, командалар тобын, командалардың шартты орындалуын сонымен қатар орындалу мен айналымдардың шартты операторларының орындалу механизмдері.

Командаларды орындаудың тізбектей және шартты орындау құралдары Linux жүйесінде анықталған құралдарға өте ұқсас. Осы құралдардың қысқаша сипаттамасы 5.3-кестеде келтірілген.

Бұл операцияларды қажет болса бір жолақта жинақтауға болады. Сонымен қатар, тек бір ғана операциялардың пайдаланылуын ғана емес, бір командаға бірнеше түрлі операцияларды біріктіруге рұқсат. Мысал ретінде төмендегі командалардың пайдаланылуын қарастыруға болады:

```
(((cd c:\logs && dir) > list.txt 2>&1) || echo Unable
to get list of logs) & start list.txt
```

Осы мысалда c:\logs каталогына ауысу әрекеті жасалады. Егер осы операция сәтті аяқталса, онда ағымдағы каталогтағы файлдар тізімін алу үшін dir команданы орындалады да осы тізім list.txt файлында жазылады. Егер операциялардың біреуі сәтсіз аяқталса, консалға «Unable to get list of logs» хабарлама шығады, ал list.txt файлына пайда болған ауытқу туралы жүйелік ақпарат жазылады. Командалар деректері жұмыстары аяқталғаннан кейін .txt (мысалы, notepad) кеңейтілуімен байланыстырылатын қосымша жіберіледі және оған list.txt файл параметр ретінде беріледі.

5.3-Кесте. Командалардың тізбектей және шартты орындалу механизмдері

Операция	Сипаттама
&	*команда1 & команда2* Бір жолақтағы бірнеше командаларды бөлу үшін қолданылады. Алдымен бірінші команда орындалады, оның жұмысы аяқталған соң екіншісі орындалады
&&	*команда1 && команда2* Бірнеше командаларды шартты орындау үшін қолданылады. Бірінші команда орындалады, егер ол сәтті аяқталса (яғни қайтару коды ретінде 0 қайтарды), онда екіншісі орындалады
	команда1 || команда2 Бірнеше командаларды шартты орындау үшін қолданылады. Бірінші команда орындалады, егер ол сәтсіз аяқталса (яғни 0-ден ерекше кодты қайтарды), онда екіншісі орындалады
()	*(команда1 & команда2)* Командалар тізбегін топтастыру үшін қолданылады
; немесе ,	*команда1 параметр1;параметр2* Командада командалық жолақта берілетін параметрлерді бөлу үшін қолданылады

Тапсырманы сипаттаудың көптеген тілдеріндегідей, Windows командалық тілінде де шартты операторлар қолданылады. Бұған қоса шартты операторларды берудің де бірнеше мүмкін формалары бар. шартты операторды жазудың жалпы формасы келесідей:

```
if өрнек команда1 [else команда2]
```

Осы жағдайда команда 1 тек егер өрнек орындалса ғана орындалады. ол шарт орындалмаса команда 2 орындалады. Сонымен бірге else шартты операторын қолданған кезде, конструкцияда else операторы if операторы орналасқан жолақта орналасқан болуы керек. Осы шарт орындалмаса интерпретатор сценарий кодында қате бар деп шығарады.

Өрнек ретінде шартты операторлардың мақсатына байланысты бірнеше конструкция қолданылуы ықтимал. Ол өрнектер тізімі 5.4-Кестеде келтірілген.

Циклды ұйымдастыру үшін командалық қабат интерфейсі Windows жалғыз ғана операторды ұсынады — for.

5.4-кесте Логикалық шарттар пішіні

Логикалық шарт	Сипаттама
[not] errorlevel *номер*	if not errorlevel 0 echo Operation is failed алдағы команда немесе операция орындалуының сәтті немесе сәтсіз болуын талдау үшін қолданылады. 0 коды көбінесе алдыңғы команда сәтті орындалғанын хабарлайды, 0 санынан басқа код көбінесе команда сәтсіздікпен аяқталды егенді білдіреді.
[not] *жолақ1 = =жолақ2*	if %1 = = %VAL% (echo Strings are the same) else (Strings are different) Екі жолақты тең/тең емес екендігін тексеруде қолданылады. Егер else операторлы пішін қолданылса, шартты оператор денесіндегі командалар () конструкциясы көмегімен топталуы керек, өйткені барлық операциялар жолақ соңында таңбамен аяқталуы шарт. Олай болмаған жағдайда интерпретатор бұл командаларды орындай алмайды және оларды елемей өте шығады.
[not] exist *файл атауы*	if exist temp.txt del temp.txt файл атауы мен файлдың бар болу/болмауын тексереді. Файл атауы жолды жазудың қатыстық түрінде сондай ақ абсолютті түрінде беріле алады.
if [/i] *1-бет Салыстыру 2-бет*	if /i %2 EQU /a echo Archive option is specified жолақтық айнымалыларды немесе командалық жолақ параметрлерін салыстыру үшін қолданылады. Салыстыру ретінде төмендегідей үш әріпті операциялар қолданылады: EQU — тең; NEQ — тең емес; LSS — қарағанда аз; LEQ — аз немесе тең; GTR — қарағанда үлкен; GEQ — үлкен немесе тең. Егер параметрі көрсетілсе, жолақтар регистр тіркеуі салыстырылады

5.4-кестесінің соңы

Логикалық шарт	Сипаттама
if cmdextversion *номер*	if cmdextversion 2 echo The operation is supported командалар кеңейтілуінің ағымдағы нұсқасын тексеру үшін қолданылады. Егер команда кеңейтілуінің ішкі номері берілген номерден үлкен немесе тең болса нақты мәнді қайтарады. Бірінші нұсқаның ішкі номері 1
if defined *айнымалы*	If defined Offset set /a MainAddr+=%Offset% айшықтандырылған айнымалының бар болуын тексеру қолданылады

Бірақ шартты оператор секілді айналым операторы оны әр түрлі мақсатта қолдануға мүмкіндік беретін әртүрлі нұсқаларға ие.

for операторы жазбасының жалпы пішіні келесідей:

```
for [параметр] {%айнымалы|%%айнымалы} in (жиынтық) do
команда [опциялар]
```

Параметр *%айнымалы* немесе *%%айнымалы жиынтықтан* элементтерді іріктеп алатын параметрді қамтиды. Жазбаның бірінші түрі егер айналым командалық жолақтан ұйымдастырылса қолданылады, ал екіншісі — егер айналым сценарий бөлігі болса.

Жиынтық элементтерінің тізімі айналымда өңделетін элементтер жинағын білдіреді. Бұл файлдар, жолақтар, мәндер диапазоны және т.б. болуы мүмкін. *Жиынтықтан* шыққан элементтер айналымда тізбектей сұрыпталады да опция ретінде командаға немесе бағдарламаларға беріледі.

Жиынтық тізіміне кіретін элементтер түрлері, опциялы for параметрмен анықталады. Параметрлер тізімі мен элемент түрлері 5.5-Кестесіне келтірілген.

Атқарым параметрлеріне командалық жолақ параметрлері үшін қолданылатын модификаторларын қолдануға болады. Мысалы, келесі мысалда ағымдағы каталогта орналасқан, dir командасымен файлдар тізімі шығарылатын форматта консолға .log кеңейтілуі бар файлдар тізімі шығарылады:

```
for %%I in (*.log) echo %%~ftzaI
```

Командалардың тізбегін өзгерту үшін сөзсіз ауысу goto операторы қолданылады:

```
goto белгі
```

Кесте-5.5. *for* айналымының элементтері	
Айналым	Элементтер түрімен сипаттамасы *for %%i in* (*.doc) *do echo %%i* Файлдар тізімін өңдеу үшін қолданылады. Файлдар тізімі тапсырмасы үшін * және ? мета таңбаларын қолданумен шаблондар қолданылады.
/D	*for /D %%dir in* (*) *do echo Directory %%i* каталогтар тізімін өңдеу үшін қолданылады. Каталогтар тізімі тапсырмасы үшін * және ? мета таңбаларын қолданумен шаблондар қолданылады.
/R *[_түбірлік каталог]*	*for /R C:\Windows %%dir in* (*) *do echo Directory %%i* Каталогтар дарағын рекурсивті тексеру үшін қолданылады, *түбірлік каталогтан* бастап. Егер каталог көрсетілмесе, ағымдағы каталогтан каталогтарды рекурсивті тексеру басталады. Тізім элементтері каталогтарда орналасқан файлдар. Егер жиынтық элементі ретінде "." элементі көрсетілсе, онда тізім файлдардың ішіндегі каталогтардан құралады
/L	*for /L %%i in (1, 1, 5) do set /a Res+ = i* Берілген диапазонда, берілген қадаммен мәнді іріктеп алу үшін қолданылады. Элемент тізімінің форматы *(басы, қадам соңы)*
/F *["параметрлер"]*	*for /F "tokens*= 1-3"» %%i *in (log.info) do @echo Time:* %%i, *Date: %%j, Event: %%k* *for /F "tokens= 1,2*" %%i in ("12.34 05.08.08 ev_3456 - String is too long") do @echo Time:* %%i, *Date:* %%j, *Event*: *%%k* *for /F "tokens*= 1,2*" %%i *in ('type log.info') do @echo Time:* %%i, *Date:* %%j, *Event: %%k тікелейс немесе басқа бағдарламадан берілетін Файлдан немесе жолақтан алынатын* жолақтар тізімін өңдеуге арналады

Осы команданы орындағаннан кейін сәйкес белгі берілген жолаққа ауысу орындалады. Егер ондай белгі табылмаса, сценарийдің орындалуы тоқтайды да белгі табылмағаны туралы ақпарат шығады.

Оператор goto сценарийдің орындалуын уақытынан бұрын үзу үшін қолданылуы мүмкін. Ол үшін келесідей жазба қолданылады:

```
goto :EOF
```

goto операторының тағы бір ерекшелігі таңдау операторын алмастыру ретінде қолданыла алатындығы.

```
goto lab%1 :lab1
echo Variant 1 goto :EOF :lab2
echo Variant 2 goto :EOF :lab3
echo Variant 3 goto :EOF
```

Бұл мысалда 1 бастап 3 дейін диапазондағы командалық жолақтан параметр қолданылады және осы параметр мәніне тәуелді сәйкес нұсқа таңдалады.

Бір командалық файлды басқасынан шақырту үшін call команданы қолданылады. Шақырту форматы келесідей:

```
call [диск:][жол]файл_атауы [параметрлер]
```

Осы кезде *диск, жол* және *файл_атауы* орындалған болуы қажет командалық файл атауын береді. Бұл файлдың кеңейтілуі .bat немесе .cmd керек. Егер қажет болса, командалық файлға параметрлер беріле алады:

```
call checkdate.bat file1.txt
```

Бірақ call команданы тек сыртқы командалар файлын іске қосу үшін ғана емес, сонымен қатар, атқарымдар мен процестерді шақыртуды ұйымдастыруда қолданылады. Ол үшін осы команданың келесі пішіні қолдданылады:

```
call :белгі [параметрлер]
```

Атқарымды шақыруды ұйымдастырған кезде: *белгі* шақырылған атқарымның басын көрсету керек. Шақырту кезінде берілетін параметрлер атқарым ішінде %1, ..., %9 позициялық параметрлерін қолдану кезінде қолжетімді.

Атқарымнан қайтару үшін /b кілті бар exit команда қолданылады:

```
exit /b [қайтару_коды]
```

Атқарымдар мен процестерді ұйымдастыру механизмі өте қуатты. Ол рекурсивті шақыртуды ұйымдастыруға көмектеседі. Процестермен жұмыс мысалы ретінде келесі сценарийді қарастыруға болады:

Fibonacci.bat:

```
@echo off
rem Командаларды шығару режимін сөндіреміз
```

```
set N=%1
call fib %N%
echo %RESULT%
exit /b

rem Фибоначчи fib сандарын есептеу атқарымы
if %1 == 1 ( set RESULT=1 exit /b )
if %1 == 2 ( set RESULT=1 exit /b )
rem санды есептейміз N-2 set /a M=%1-2
rem фибоначчи санын шығарамыз, номер N-2 call fib %M%
rem номер N-2 фибоначчи шығарылған санын стекте
сақтаймыз
set /a M=%1-2
set RES%M%=%RESULT%
rem Фибоначчи номер N-1 санын есептеп шығарамыз set /a
M=%1-1 call fib %M%
rem стектен номері N-2 фибоначчи санын шығарамыз, N-1
номерлі фибоначчи санымен қосамыз және нәтижесін  RESULT
айнымалысына жазамыз set /a M=%1-2
for /F %%i in ('echo %%RES%M%%%') do set /a RESULT+=%%i

exit /b
```

Осы орындалу үстіндегі сандық параметрі бар файлды шақырған кезде экранға оның факториалына сәйкес келетін сан шығады:

```
C:\>Fibonacci.bat 8 21
```

Өкінішке орай рекурсивті шақыртуларды ұйымдастырған кезде айнымалылардың мәндері сақталмайды. Бұл мәндер бағдарламалаушылардың өздерінің ұйымдастыруларына алып келеді. Сол себепті жоғарыда көрсетілген мысал Фибоначчи сандарын классикалық үлгіде орындаудан ерекшеленеді.

Мысалда бұдан бұрын алынған аралық нәтижелерді сақтау үшін айнымалылар стегі ұйымдастырылады (set RES%M% = %RESULT% командасы). %M% саны стектің ағымдағы деңгейін береді. Одан кейін команданың шығуын алатын және стектен мәндерді алып тастау үшін оның айнымалыларын жазатын for операторы қолданылады. Осы мәндер Фибоначчи сандар есептеуде қолданылады.

5.9.6. PowerShell командалық қабаты

Windows операциялық жүйелерінің командалық жолақ құралдары UNIX/Linux- жүйелерінің түрлі қабаттарында ұсынылатын құралдарға қуаты және ыңғайлылығы жағынан артта қалады. Microsoft фирмасының өңдеушілері үнемі командалық интерпретатордың бар функционалдығын арттырып отырады және жаңа құралдар қосады, бірақ бұл командалық сценарий құру мәселесін түбегейлі өзгертпейді—олардың зерттемесі әлі күнге дейін айтарлықтай ауыр және ыңғайсыз.

Осыған байланысты Microsoft фирмасы командалық жолақ интерфейсі және сценарийлерді құруға арналған кіріктірілген тілі бар жаңа қабат шығару туралы шешім қабылдады. Бұл қабат PowerShell деген атауға ие болды. Бұл қабат Cmd.exe/Command.com, BASH, WSH, Perl және басқа да қабаттардың қасиеттерін дамытады. Бұл қабат платформамен біріктірілген. .NET Framework мына операциялық жүйелерде пайдалануға болады: Windows XP SP3, Windows Server 2003 SP2, Windows Vista SP1, Windows Server 2008, Windows 7 және Windows 8.

PowerShell COM және WMI толық қолжетімділікті ұсынады, ол әкімшілерге жергілікті және алыс жүйелерді басқаруға мүмкіндік береді.

PowerShell ішіндегі әр команда *командлет* деп аталады, ол берілген операцияны орындайтын арнайы классын .NET қамтиды. Бірнеше командлеттерді бір сценарийге немесе тәуелсіз орындалатын бинарлы бағдарламаға біріктіруге болады.

Windows PowerShell сырттан келген қосымшаларға PowerShell командлеттерін қолдануға мүмкіндік беретін арнайы механизм ұсынады. Мысалы, Microsoft Exchange Server 2007 осы механизмді PowerShell ортасында әкімшілерге басқарудың өз механизмін ұсыну үшін қолданады. Оған қоса, жүйе әкімшісінде деректер базасының күйімен командлеттер интерфейсі арқылы басқару мүмкіндігі туады.

PowerShell төрт түрлі команданы орындай алады:

1) командлет .NET жабын ядросымен жүктелетін және орындалатын арнайы кітапхана түрінде беріледі;
2) PowerShell сценарийлері (.ps1 кеңейтілуі бар);
3) PowerShell атқарымдары;
4) бинарлық бағдарламалар.

PowerShell командаңың мысалы ретінде келесі мысалдарды қарастыруға болады.

Барлық алдын-ала анықталған командлеттерді қарау үшін Get-Command командасын енгізу жеткілікті. PowerShell бойынша анықтама алу үшін Get-Help командасын енгізу қажет.

P таңбасынан басталатын барлық процестерді аяқтау үшін келесі команданы енгізу жеткілікті:

```
PS> Get-Process p* | Stop-Process
```

Қандай процес аяқталғанын тексеру үшін немесе процес тоқтағанша сценарийдің орындалуын тоқтату үшін келесі команданы енгізеді:

```
PS> $processToWatch = Get-Process Notepad
PS> $processToWatch.WaitForExit()
```

Синонимдер механизмін пайдаланып, бұл сценарийді әлдеқайда қысқартып жазуға болады:

```
PS> (ps notepad).WaitForExit()
```

Get-Process командлетінде ps синонимі бар. Осы синонимді пайдаланып, сценарийді қысқартуға болады. Көптеген синонимдер толықтай осы жабындағы командалармен ұқсас болғандықтан, оларды BASH сценарийіне ұқсас етуге болады. Барлық синонимдер тізімін алу үшін Get-Alias командлетін орындау жеткілікті.

.NET Framework негізінде PowerShell құрылғандықтан және оның көптеген қасиеттерін иеленетіндіктен, .NET тәсілдерін және класстарын қолдану мүмкіндігі пайда болады. Мысалы, сценарийде қандай да бір саннан түбір табу қажет болса, ол үшін Sqrt() статикалық тәсілін шақыруға болады:

```
PS> [System.Math]::Sqrt(16) 4
```

Жалпы айтқанда бұл жабын әкімшілердің мүмкіндіктерін айтарлықтай кеңейтеді, .NET барлық қуатын пайдаланып, анағұрлым оңай сценарийлерді жазу мүмкіндігін береді және графикалық жабынды қолданбай-ақ жүйенің барлық қызметтеріне қолжетімділікке ие болады, әдетте жасалатын әрекеттер үшін өте ыңғайлы.

Б
А
ҚЫЛАУ СҰРАҚТАРЫ

1. Операциялық жүйедегі тапсырманы сипаттау тіліне қандай қасиеттер тән?
2. Операциялық режимдегі диалогтық және пакеттік режимдердің айырмашылығы неде?
3. Орындалу үстіндегі тапсырмаға командалық жолақтың параметрлері қалай беріледі?
4. Командалық жолақтың тоғыз және одан көп параметрлеріне қалай қол жеткізуге болады?
5. Енгізу/шығару, қайта бағыттаудың қандай құралдары операциялық жүйе арқылы қамтамасыз етіледі?
6. Тапсырма айнымалыларын тапсырманы орындау ортасына көшіру қалай орындалады? Қандай шектеулер бар?
7. Пайдаланушы тапсырманы орындау тілінде қалай бағдарлама жасай алады?

Тарау 6

ЖҮЙЕ ПАЙДАЛАНУШЫЛАРЫ

6.1. ЖҮЙЕ ПАЙДАЛАНУШЫЛАРЫ ЖҮЙЕГЕ КІРУ

Алдыңғы тарауда, пайдаланушыға қолжетімді, операциялық жүйемен өзара әрекеттесетін негізгі тәсілдер көрсетілген. Осы ретте, осы кітапта негізгі қаралатын - UNIX операциялық жүйесі - көп пайдаланушы екендігін ұмытпаған жөн. Осы тарауда көп пайдаланушыны сүйемелдеу қалай ұсынылатыны, яғни негізінен, сеанстарды басқару мен ақпараттарды қорғау қозғалады.

UNIX-жүйесімен жұмыс сеансын бастау үшін өзіңіздің есептікке алу деректеріңізді — *жүйеге кіру шақыртуына* жауап ретінде пайдаланушының аты мен паролін енгізу қажет. Жүйеге кіру шақыртуы әдетте, жүйенің жүктелімі аяқталған соң, терминалда пайда болады. UNIX орнатылған серверге қосылу *telnet* терминалының желілік хаттамасы арқылы жүргізіледі. Мұндай желілік қосылу кезінде пайдаланушы компьютерінің экранына, сервермен қызмет көрсетілетін логикалық терминалдың бірінің экранынан ақпараттар жіберіледі, ал пайдаланушының пернетақтамен енгізген ақпараттары, серверге жіберіледі.

Екі жағдайда да, жүйеге кіруге шақырту шамамен осылай көрінеді:

login:

Бұл шақыртуға жауап ретінде, өзіңіздің есептікке алу деректеріңізді енгізу талап етіледі, осыдан кейін экранға пароль сұрау салу шығады.

password:

Егер пайдаланушымен енгізілген есептікке алу деректері (логин) мен пароль дұрыс болса, онда жүйеге кіру жүргізіледі және негізгі командалық интерпретатор іске асырылады, яғни пайдаланушының жүйедегі жұмыс сеансы кезіндегі белсенді болып табылатын, командалық интерпретатор. Бұл командалық интерпретатордың аяқталуы, пайдаланушы жұмыс сеансын аяқтайды. Әдеттегідей жүйелердің көбісі, пайдаланушылар үшін BASH командалық интерпретаторы колданылады.

Егер жүйенің пайдаланушысы адам болып табылса, жоғарыда қарастырылғандардың бәрі әділетті. Дегенмен бұл әрқашан дұрыс болмайды - жүйенің пайдаланушысы ретінде, операциялық жүйе ұсынатын, обьектіні пайдалану құқығы бар, кез келген обьекті болуы мүмкін. Мұндай құқықтарға үлгі болып, нақты файлдарға қол жеткізу құқықтары қызмет етеді, бағдарламалар орындалуының іске қосылуы, басып шығару құрылғыларына қол жеткізу.

Пайдаланушылардың рөлінде, белгілі бір құқықтар жинағымен іске асырылатын бағдарламалар болуы мүмкін. Мұндай бағдарламалар нақты пайдаланушының атымен іске асырылады, мұндай жағдайда олар *псевдопайдаланушы* деп аталады. Мысалы, UNIX- жүйедегі типті псевдопайдаланушы — mail есептікке алу деректері бар пайдаланушы. Бұл пайдаланушының атына, электронды пошталардың хабарламалар кезегін басқару, хабарламаларды жеткізу және мекенжайларға тасымалдау жүргізіледі. Қарапайым пайдаланушы, әдетте псевдопайдаланушы атымен жүйеге кіре алмайды, себебі жүйеге кіру шақыртуының көмегімен қолжеткізу мүмкіндігі, псевдопайдаланушылар үшін блокталған.

Қарапайым пайдаланушылар мен псевдопайдаланушылардан басқа, жүйеде жүйе әкімшісідің қызметін атқаратын, ең кемі бір *суперпайдаланушы* бар. Бұл пайдаланушыда операциялық жүйемен ұсынылатын, барлық ресурстарға қолжеткізу мүмкіндігі бар. UNIX-жүйелерінде бұл пайдаланушы әдетте root есептікке алу деректерге ие.

6.2. ПАЙДАЛАНУШЫЛАРДЫҢ ҮЙ КАТАЛОГЫ

Операциялық жүйеде жұмыс істейтін әр пайдаланушыға жеке файлдарын сақтайтын және жұмыс жасауға арналған каталогтар бөлінеді. Мұндай каталог дәстүрлі үй каталогы деп аталады. Әдетте үй каталогтары пайдаланушының логиніне сәйкес келетін атауларға ие және /home каталогында орналасқан. Мысалы sergey пайдаланушысы үшін үй каталогы әдетте /home/sergey атауына ие.

Үй каталогын атаулы жолмен көрсету үшін ~ мнемоникасы қолданылады. Мысалы өзіңіздің үй каталогыңызға өту үшін команданы енгізу қажет. Үй каталогының dirname шағын каталогына өту үшін

```
cd ~/dirname
```

командасын енгізу қажет.

Пайдаланушының белгілі логинмен үй каталогына өту үшін ~<login name> мнемоникасы қолданылады. Мысалы, nick пайдаланушысының үй каталогына өту үшін

```
cd ~nick
```

енгізуі талап етіледі.

Осы ретте «бөгде» үй каталогына өту, осы каталогқа қолжеткізу құқықтары жеткілікті болғанда ғана жүзеге асады (6.4 тармағын қараңыз).

Бұдан басқа, үй каталогының атауы дәстүр бойынша, $HOME айнымалы шеңберінде тұрады. Сәйкесінше, кез келген уақытта

```
cd $HOME
```

командасының көмегімен, өзінің үй каталогына өтуге болады.

Бұл айнымалының мәнін BASH құралдарымен өзгертуге болады, бірақ былай жасауға кеңес берілмейді —UNIX жүйелік утилиттардың көбісі бұл айнымалының мәнін, бұл бағдарламалардың баптауларымен бірге файлдарды сақтау үшін мүмкін болатын жолдарды анықтау үшін қолданады. Егер $HOME айнымалысында еркін мәндері көрсетілсе, баптауларды сақтау мүмкін емес болады, ал кейбір бағдарламалар жұмысқа қабілетсіз болып қалады.

6.3. ПАЙДАЛАНУШЫЛАРДЫ СӘЙКЕСТЕНДІРУ

Жүйенің әр пайдаланушысы берілген жүйенің шегінде бірегей есептікке алу деректеріне (логин) ие болады. Бірақ, бұл атау тек пайдаланушының ыңғайлылығы үшін тағайындалады — операциялық жүйе пайдаланушыларды бірегей *идентификаторы бойынша* айырады — UID (User IDentifier). UID идентификаторы бүтін санды, үлкен немесе нөлге тең санды ұсынады. UID, пайдаланушының есепке алу аты сияқты бірегей.

Одан басқа, әр пайдаланушы үшін UID және логин, пароль, паспорттық атауы, үй каталогына жол, әдеттегідей командалық интерпретатордың орындалатын файлының толық атауынан тұратын

жүйелік және анықтамалық ақпараттарды, атрибуттар жинағын анықтайды. Бұл ақпараттар /etc/passwd файлында сақталады. Әр пайдаланушы туралы ақпарат жеке жолда тұрады, атрибуттары «:» қос нүктемен бөлінген. Атрибуттардың сатылығы мынадай: тіркеме атауы, пароль (қысқартылған түрде) 6 пайдаланушының идентификаторы (UID), топтардың идентификаторы (GID), паспорттық атауы, үй каталогына жол және командалық интерпретатордың толық атауы. Мысалы, UID 1001 ие пайдаланушы Вася Пупкин, /etc/passwd файлында келесі жолмен ұсынылады:

```
vasya:Fes8s9xapl:1001:10:Vasya Pupkin:/home/vasya:/bin/bash
```

Пайдаланушы әрқашан бір немесе бірнеше пайдаланушылар тобының мүшесі болып табылады. Егер де пайдаланушы, жүйеге қолжетімділігі бар тек бір адам болса, онда ол минимум «Жүйенің пайдаланушылары» (әдетте users атауымен) немесе «Әкімгерлер» тобының (әдетте root атауымен) мүшесі болып табылады.

Пайдаланушылар тобы — тізім ретінде берілетін, пайдаланушылардың көпшілігі. Пайдаланушылардың топқа бірігуі, әдетте пайдаланушылармен орындалатын міндеттерді шектемеу принципі бойынша жүргізіледі. Осылай, жеке топтарға жүйенің әкімгерлері бөлінеді.

«Білімді бақылау» жүйесінде пайдаланушылардың келесі топтарын бөлуге болады: жүйенің өңдеушілері, студент, оқытушы. Осылай топтарға бөліп, бірінші кезекте пайдаланушылардың жүйеге рұқсатының әр түрлі деңгейлеріне ие болуымен байланысты.

Осылай, өңдеуші, жүйенің негізін құрайтын тапсырмаларын жаңалап отыруға және жүйенің каталогтар құрылымын жетілдіре алатын мүмкіндікке ие болуы керек; оқытушы бақылау жұмыстарының базасын жаңалауға және орындалған жұмыстарды тексеретін құқыққа ие болуы керек, ал студент — алынған жұмыстарды қарау және орындау мүмкіндігіне ие болады.

Жүйенің жұмыс каталогтарының атауларына сәйкес келетін, жүйенің өңдеушісі devel есептікке алу деректердің, оқытушы— teacher, студенттер — ерікті атауға ие болады. Одан басқа, teacher және devel - кейбір каталогтарға студенттердің қол жеткізу құқығын шектейтін teacher тобына кіреді.

Әр топ бірегей есептікке алу деректері мен GID тобының бірегей идентификаторына (Group IDentifier) ие болады..

Жүйеде анықталған топтар туралы ақпараттар, /etc/group файлында сақталады. Әр топ туралы ақпарат жеке жолда орналасады, топтардың атрибуттары «:» қос нүкте символдарымен бөлінген. Атрибуттардың сатылығы мынадай: топтардың тіркеме атаулары, топтардың жағдайларының жалауы (әдетте «*» символы), топтардың

идентификаторы (GID), үтірмен белгіленетін, топтарға кіретін қолданушылардың тізімі. Файл жолының үлгісі /etc/group төменде көрсетілген:

```
users:*:1001:sergey,nick,alex
```

6.4.

ФАЙЛДАР МЕН КАТАЛОГТАРҒА ҚОЛ ЖЕТКІЗУ ҚҰҚЫҒЫ

Көптеген операциялық жүйелерде файлдарға шектеу құралдары бар. Бір пайдаланылатын ОЖ-де әдетте, операциялық жүйелердің ядролық файлдарына қолжетімділік шектеледі (мысалы, DOS «жасырын» атрибутын орнату).

Көп қолданылатын ОЖ қолжетімділік осы немесе басқа пайдаланушының файлға қолжетімділік құқығын анықтау көмегімен шектеледі. Қолжетімділік құқығы файлға сол немесе басқа операциялардың орындалу мүмкіндігін анықтайды. Жалпы айтқанда, «файлға қолжетімділік құқығы» термині дұрыс емес. Бұл ақпараттар жиынтығымен байланысты барлық файлдар үшін де, сол ақпараттар жиынтығына арналған қолжетімділік құқығы анықталған. Бір ақпарат жиынтығына байланысты барлық файлдар үшін бұл құқықтар бірдей.

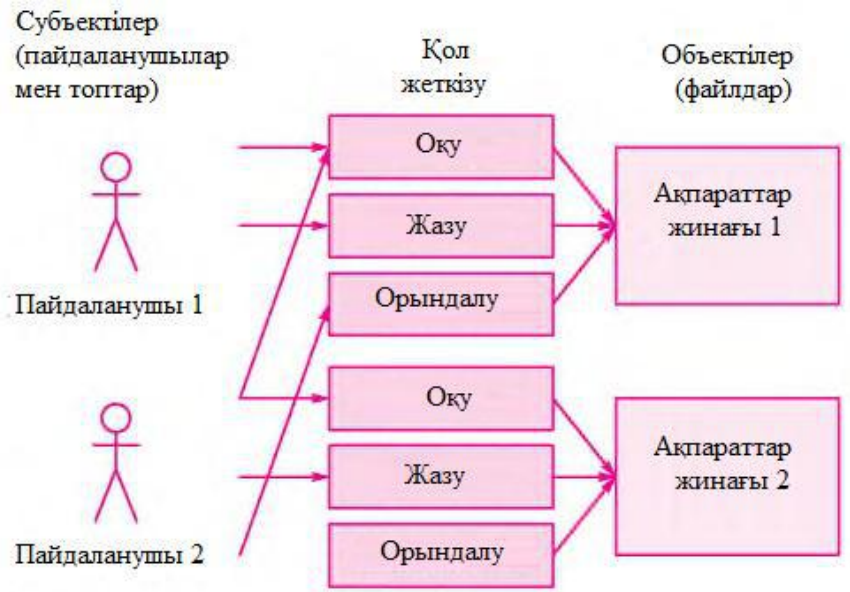
Пайдаланушылардың файлға қолжетімділігін үш элементтер арқылы – субъект-обьект- қолжетімділік құқығымен анықталады. Сонымен қатар субъект деп обьектіге қарайтын обьект түсіндіріледі немесе нақты көп пайдаланушы – обьект ретінде – файл немесе каталог, ал қолжетімділік құқығы – обьектіге сол немесе басқа операциялардың орындалуына субъектінің қолжетімділігі немесе тыйым салуы (сурет 6.1).

UNIX-жүйелерде субъектінің құқығын белгілейтін үш түр бар: файл иесі немесе пайдаланушы –иесі(u), пайдаланушы тобының файлды игеруі немесе ие тобы (g), және жүйелердің барлық пайдаланушылары (o).

Әр субъект түрі үшін операцияның үш түрінің орындалуына тыйым салуға немесе қолжетімділікті беруге болады: оқу (r), жазу (w) немесе орындалуы (x).

Файлды оқу құқығында осы файлды ашуға және ондағы ақпаратты оқуға мүмкіндік беріледі. Жазу құқығында – файлдағы ақпаратты өзгерту немесе файлды өшіру мүмкіндігі бар. Орындалу құқығында осы файлдың құрамындағы бағдарламаны іске асыру мүмкіндігі тұспалданып тұр (сонымен қатар орындалу процесі пайда болады).

UNIX-жүйесінде әр файлдың екі иесі бар: пайдаланушы-иесі және топ-иелері. Сонымен қатар пайдаланушы иесі топ иелерінің мүшесі болуы міндетті емес, бұл әр түрлі пайдаланушылар үшін файлдар мен каталогтарға қолжетімділікті оңтайлы басқаруға мүмкіндік береді.



6.1- сурет. Субьект—объект—қол жеткізу құқығы үштігінің көмегімен қол жеткізу құқығын беру

```
drwxrwx---  4  nikita users      0 Apr 28 23:25 texmf 375
-rwxr-x---  1  nikita users  Apr 26 04:27 textmode.o 452
-rwxr-x---  1  nikita users  Apr 26 04:27 readmode.o 0 Apr
drwxrwx---  2  nikita users  28 23:22 tk8.4 2948 Sep 2
-rwxr-x---  1  nikita users  20:03 tkConfig.sh
```

Файлдар мен каталогтар иелері туралы ақпарат алу үшін l командаңың кілті ls. ls -l командасы ағымдағы каталогтағы файл туралы ақпаратты мына түрлері бойынша шығарады:

Үшінші бағанда файлдың пайдаланушы-иесінің тіркелген атын, ал төртіншіде – топ-иелерінің тіркелген атын шығарады. Бірінші бағанда файлға қолжетімділік құқығын беретін файл атрибуттары туралы ақпаратты құрайды.

Бағанадағы бірінші символ файлдың түрін береді. Каталогтар үшін бірінші символ d мәнін, ал қарапайым файлдар үшін — сызықты білдіреді. Басқа да файл түрлері бар.

Екіншіден оныншыға дейін символдар үштік символдарға бөлінген (триады), олардың әрқайсысы нақты субьектінің қолжетімділік құқығын анықтайды. Бірінші триада файлдың пайдаланушы - иесінің,

екінші триада – пайдаланушы-топтың, ал үшінші триада – жүйенің пайдаланушылар үшін құқығын анықтайды.

Солай, textmode.o файлы үшін пайдаланушы- иесінің файлға толық жеткізу құқығына (яғни, оқу, жазу, орындалуға құқығына ие), пайдаланушылар – тобы оқуға және орындалу құқығына ие, ал жүйені пайдаланушылардың қалғаны бұл файлға ешқандай қолжетімділік құқығына ие болмайды.

R, w және x атрибуттарының файлға іс-әрекеті жеткілікті көзге көрінерлік — бұл атрибуттар файлдар үшін Open, Read және Write жүйелік шақырылымдардың орындалу мүмкіндігін анықтайды. Файлды оқу құқығы болса, пайдаланушының бағдарламасы бұл файлды Open шақырылымымен аша алады және ақпараттарды Read шақырылымен оқи алады. Сондай-ақ, жазу құқығы болса, пайдаланушы бағдарламасы Open және Write шақырылымдарын қолданады. Бағдарламаның орындалатын файлында орнатылған бағдарламаның орындалуына құқық, пайдаланушыға бағдарламаны басқаруды береді, сонымен қатар операциялық жүйе Open және Read көмегімен, бағдарламаның орындалатын кодын оперативті жадығы береді және оған басқаруды жібереді.

Каталогтар үшін r, w және x атрибуттары енді көзге көрінерліктей емес, және бірінші ретте каталогтар үшін Open, Read и Write жүйелік шақырулар ерекшеліктерімен байланысты.

Каталог үшін r қолжетімділік құқығы, каталогта сақталған файлдардың атын оқуға мүмкіндік береді. Бұл файлдарға қолжетімділік мүмкіндігі файлдарға қолжетімділік құқығымен анықталады.

Каталогтар үшін x қолжетімділік құқығы метаақпарат блогында сақталған каталог файлдары барлық атрибуттарын алуға мүмкіндік береді. Егер x құқығы каталогтар және кіші каталог үшін орнатылған болса, онда ол берілген каталогқа өтуге мүмкіндік береді, яғни оны ағымдағы етеді.

Каталогтағы жазу мүмкіндігі каталог мазмұнын өзгертуге мүмкіндік береді, яғни осы каталогтағы файлдардың пайда болуына және өшірілуіне мүмкіндік береді. Мұнда, каталогтағы жазу құқығы бар болса, оған дейін қолжетімділік құқығы болмаған файлдарды өшіру мүмкіндігі бар екенін айта кету керек. Егер Unlink жүйелік шақыруының әрекетін еске түсірсек, жүйенің бұл күйін түсіну оңай — ол ақпарат жинағында қатқыл сілтемені, ақпарат жинағының өзін өзгертпей-ақ, өшіреді. Бұның ізінше, әсіресе каталогтағы жазуға қолжетімділік құқығын тағайындағанда сақ болу керек.

6.4.1. Файлдар Мен Каталогтарға Қолжетімділік Құқығын Беру

Файлдар мен каталогтарға қолжетімділік құқығы уақыт ағынымен өзгеруі мүмкін. Бұл мысалы, файлдың бұрынғы иесі оны өзгерту құқығын жаңа иесіне бергенде немесе бұрын бір пайдаланушыға ғана қолжетімді файл, бәріне бірдей қолжетімді болуымен байланысты болуы мүмкін.

Файлдың қолжетімділік құқығын өзгертуге үлгі болып, уақыт ағынымен бақылау жұмысының нұсқасына қолжетімділік құқығын өзгертуге қызмет етуі мүмкін. Толтырылмаған бақылау жұмысының үлгісі оқытушының жалпы базасында болғанша, ол тек оқытушыға қолжетімді, нұсқалы файл студентке берілгенде ғана оған қолжетімді болады.

Студентке бұл файлға кіру құқығын беру үшін иесін өзгерту керек немесе бұл файлға толық рұқсатты ашу керек, мысалы мынадай:

```
cp /check/teacher/theme1/var2.txt \
/check/students/vasya/theme1_var2.txt
chmod 666 /check/students/vasya/theme1_var2.txt
```

Ие-пайдаланушы немесе пайдаланушылар тобы chown (CHange OWNer) командасының көмегімен өзгеруі мүмкін. Файлдың иесін UNIX-жүйесінің көбісінде тек жүйе әкімшісі ғана өзгерте алады.

chown командасының бірінші параметрі пайдаланушының тіркеу атын және топтың тіркеу атын анықтайды. Екінші және келесі параметрлері, иелері өзгеретін файлдар тізімін құрайды.

Егер бірінші параметр пайдаланушының атын және топтың атын анықтаса, онда олар нүктемен бөлінеді. Мысалы, команда

```
chown nick file.txt
```

nick пайдаланушысын file.txt файлының пайдаланушы- иесі етеді. Ал команда

```
chown nick.admins file.txt
```

nick пайдаланушысын пайдаланушы- ие етеді, admins — тобын file.txt файлының пайдаланушылар-тобы етеді.

Файлдар мен каталогтарға қолжетімділік құқығы П chmod (CHange MODe) командасы көмегімен өзгеруі мүмкін. Қолжетімділік құқығын тек файлдың иесі немесе жүйе әкімшісі өзгерте алады. Дәлел ретінде chmod командасына, қолжетімділік анықталатын, файлдар аты мен қолжетімділіктің спецификаторлары беріледі.

Қолжетімділік спецификаторы — файл қолжетімділік құқығын анықтайтын символдар жолы. Спецификатор бірінен кейін бірі тұратын элементтерді құрайды: субъект, қолжетімділік құқығы және қолжетімділік құқығы операциялары. Субъектті беру үшін: «u» — пайдаланушы-иесінің символы көрсетіледі, «g» — пайдаланушылар тобы, «o» — барлық қалған пайдаланушылар, «a» — алдыңғы үш субъектілер. Қолжетімділік құқығы ретінде «r» — оқу, «w» — жазу және «x» — орындалу символдары көрсетіледі. Қолжетімділік құқығы операциялары, субъектіге арналған қолжетімділік құқығы жинағын өзгертеді. «-» операциясы қолжетімділік құқығын алып тастайды, «+» қолжетімділік құқығын қосады, «=» субъектіге нақты берілген қолжетімділік құқығының жинағын тағайындайды.

Қолжетімділіктің қарапайым спецификаторлы chmod мынадай көрсетіледі:

```
chmod a+w file.txt # жазуға бәріне рұқсат
беру

 # файл file.txt
```

Chmod командасына берілетін қолжетімділік спецификаторлары, мынадай үтір арқылы аталады:

```
chmod u+w,g=r file.txt #иесіне қолжетімділік ету-
                        #пайдаланушыға жазу

                       #пайдаланушылар-тобына
қолжетімділік ету                              #тек
file.txt  файлынан оқу
```

Бір қолжетімділік спецификаторында бірнеше субъектілер берілуі мүмкін: `chmod ug+w file.txt #пайдаланушы-иесіне қолжетімділік ету`

```
                     #пайдаланушы-тобына     қолжетімділік
ету
#және пайдаланушы - тобына file.txt файлына жазу
```

Сондай ақ, бір спецификаторда субъект үшін бірнеше операциялар және қолжетімділік көрсетілуі мүмкін:

```
chmod u+w+r file . txt #пайдаланушы-иесіне қолжетімділік
ету
                      # file.txt файлында оқу және жазу
```

Файлға қолжетімділік құқығын берудің баламалы тәсілі— сандық. Қарастырылған тәсілден өтуді түсіндіру үшін барлық үштіктерді

жазамыз:

```
rwx rwx rwx
```

қолжетімділік құқығына екілік бірлікті және екілік нөлге – орын жоқтығын қабылдаймыз. Сонда әр үштік - 000 ден 111 ге дейін екілік санмен анықталуы мүмкін. Солай, r—x үштігі 101 екілік санына сәйкес келеді. Әр екілік сан, одан кейін саналудың сегіздік жүйесіне ауыстырылуы мүмкін.

Үштің осылайша ауыстырылуының процесінде

`rwx rw- r-` аламыз.

111 110 100

Немесе `764`

Файлға қолжетімділік құқығын сандық ретінде ұсыну, сондай ақ chmod командасына қолжетімділік спецификаторы түрінде қолданылуы мүмкін. Осылайша,

`chmod 7 64 file.txt` командасы

пайдаланушы-иесіне толық рұқсатты береді, пайдаланушылар- тобына оқуды, жазуды және барлық қалған пайдаланушыларға – тек оқуға рұқсатты береді.

«Білімді бақылау» жүйесінде, каталогтар жүйесіне қолжетімділікті, оның пайдаланушыларымен болатын, жүйенің бөліктеріне зақымдануды болдырмас үшін осылайша орнату керек. Одан басқа, жалпы базадағы бақылау жұмыстарының нұсқасына студенттердің қолжетімділігіне, сондай ақ көшіріп алуды (студенттердің басқаның жұмысына қолжетімділігі) болдырмау. Оқытушылардың тапсырмаларды беруін (оны студент каталогына көшіру) және жұмыстың тапсырылуын (оны орындалған жұмыстар каталогынан орын ауыстыруы) қамтамасыз ету керек.

Пайдаланушылардың жүйенің негізгі құрауыштарына (сценарийына және каталогына) қолжетімділікті тыю үшін олардың иелеріне devel пайдаланушысы (жүйені өңдеуші) тағайындалады:

chown devel.teacher /check/scripts

сонымен қатар берілген каталог орнатылған

құқықтармен шектеледі:

```
chmod 755 /check/scripts
```

басқаша айтқанда, барлық пайдаланушылар сценарийдың орындалуына және оқуына қолжетімділік алады, бірақ тек өңдеуші ғана жазуға қолжетімділік ала алады.

Devel пайдаланушысы, құрамына оқытушы (teacher логинімен пайдаланушы) кіретін teacher (оқытушылар) тобының мүшесі болып табылады. Бұл топ, студенттердің бақылау жұмыстары каталогына қолжетімділігін шектеу үшін енгізілген.

```
chown $i.teacher /check/students/vasya
chmod 730 /check/students/vasya
chown $i.teacher /check/students/vasya/ready
chmod 770 /check/students/vasya/ready
```

6.4.2. Файлдар мен каталогтарға қол жеткізу құқығын тексеру

Файлдар мен каталогтарға қолжеткізу құқығын шектеу, пайдаланушының тапсырмасы орындалатын ақпараттық ортаны өзгертеді. Егер пайдаланушы тапсырма жұмысына әсер ететін, яғни тапсырмалардың ақпараттық ортасына кіретін барлық файлдарға қол жеткізу құқығына ие болса, алаңдайтын ештеңе жоқ. Бірақ қажетті файлдар немесе каталогтардың талап етілген қолжетімділік режимінде қолжетімсіз болатын жағдайлар болуы мүмкін. Сондықтан тапсырмаларда ондай жағдайларды болдырмау үшін өңдеуді алдын ала қарастыру керек. Әдетте, өңдеушілер, болдырмайтын жағдайларды тапсырманы мәтінінің басына ауыстырады және ақпараттық ортаның қолданылуын тапсырмамен тексереді. Осы жағдайда қолдану мүмкіндігі деп - қажетті қолжетімділік құқығының болуымен түсіндіріледі.

Файлға қолжетімділік мүмкіндігін тексеруге арналған негізгі параметрлер 5 тарауда қарастырылған test командасы қолданылады. Test командасы тапсырма тексерілетін тұлғадан (ағымдағы пайдаланушы), пайдаланушыда қолжетімділік құқығы бар болуын

тексереді. Ол үшін команданың келесі параметрлері пайдаланылады:

- -r <файл> — ағымдағы пайдаланушыға тек файлды оқу құқығы рұқсат етіледі;
- -w <файл> — ағымдағы пайдаланушыға тек файлға жазу қолжетімділігі рұқсат етілген;
- -x <файл> — ағымдағы пайдаланушыға файлдың орындалуына қолжетімділік рұқсат етілен.

Мысалы, outfile.txt файлына жазу құқығы барын және infile.txt, файлына оқу құқығы барын тексеру үшін BASH тілінде тапсырмалардың келесі фрагментін орындау жеткілікті:

```
if [ ! -w outfile.txt -a ! -r infile.txt ] ;
then echo "Insufficient access rights" exit 1
fi
```

Қосылған мұндай фрагментте, тапсырмалардың қолжетімділік құқығы жетіспеген жағдайда экранға «Insufficient access rights» хабарламасы шығады және өзінің орындалуын 1 қайтару кодымен аяқтайды.

Б
ҚЫЛАУ СҰРАҚТАРЫ

1. Операциялық жүйеде қандай параметрлер пайдаланушыны сипаттайды?
2. Пайдаланушының жұмыс сеансы қалай басталады? Неліктен жұмыс сенасын түзу аяқтау маңызды?
3. Операциялық жүйелерде файлдар мен каталогтарға қандай шектеулер ұсынылады?
4. Файлдар мен каталогтарға қатысты пайдаланушының қолжетімділік құқықтары қалай беріледі, өзгереді, тексеріледі?
5. Файлдар мен каталогтардағы қолжетімділік құқықтарының айырмашылықтары неде?
6. Оқу үшін қолжетімді емес және пайдаланушылардың үй каталогында болатын, файлдар тізімін шығаратын командалық файлды жаз.

7 ТАРАУ

ПАЙДАЛАНУШЫЛАРДЫҢ ФАЙЛЫ

7.1. UNIX ЖӘНЕ WINDOWS КАТАЛОГТАРЫНДА ЖҮЙЕНІҢ СТАНДАРТТЫ ҚҰРЫЛЫМЫ

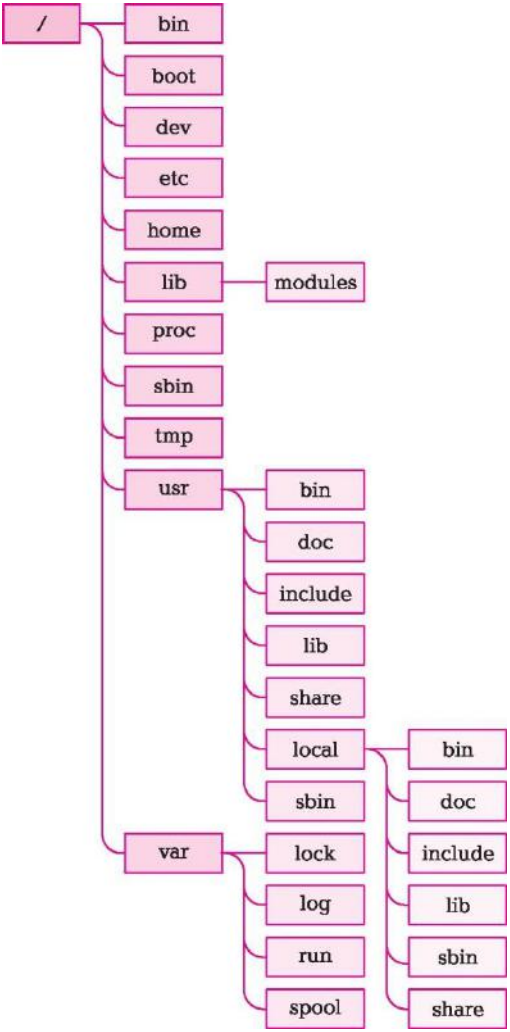
UNIX сияқты операциялық жүйелерде жеке ақпараттарды сақтау үшін 7.1. суретте көрсетілген каталогтардың стандартты құрылымы пайдаланылады.

Нақты операциялық жүйені іске асыруға байланысты келтірілген құрылымнан кейбір айырмашылықтар болуы мүмкін, мысалы SunOS және Solaris операциялық жүйелерде /opt каталогы қатысады.

/Bin каталогы базалық жүйелік утилиттерін сақтау үшін қызмет етеді. Әдетте, бұл утилиттер негізгі жүйелік кітапханаларда болмаған жағдай да орындалуы мүмкін. /Boot каталогтарында жүктелімге қажетті ақпараттар мен операциялық жүйенің ядро файлдары сақталады. /Dev каталогы құрылғыларға (7.2 бөлімшені қараңыз) қаратулар болуының көмегімен арнайы файлдарды құрайды. Бұл каталогта болатын әр файл, қандай да бір файлға сәйкес келеді. Файлға ақпараттарды жазу, осы ақпараттарды құрылғыға берілуін шақырады, ал оқу – құрылғыдан ақпараттарды оқуды /Etc каталогы баптау файлдарын және файлдардың пішін үйлесімін сақтауға арналған. /Home каталогында пайдаланушылардың үй каталогтары тұрады (келесі тарауды қараңыз). /Lib каталогы жүйелік кітапханаларды сақтау үшін ал /lib/modules — жүйешелер мен драйвер ядроларының жүктелімі аяқталған соң, операциялық жүйенің ядролар модулін сақтауға арналған. /Proc каталогы құрылғылардың параметрлерін қарау және ОЖ ядро кестелеріне қолжетімділік мүмкін болатын, арнайы файлдарды құрайды. /Sbin каталогы жүйе әкімшісіне арналған жүйелік утилиттерді сақтайды. /Tmp уақытша файлдарды құрайды.

/Usr каталогында жүйелікке қатысы жоқ және операциялық жүйе қондырмасының жинағына кіретін бағдарламалық қамтамасыз етуді құрайды. /Usr/bin, /usr/sbin, /usr/lib каталогтары жоғары деңгейдің сәйкес каталогтары тәрізді. /Usr/doc каталогында ОЖ жеткізілім жинағына кіретін бағдарламалық қамтамасыз ету құжаттары сақталады, /usr/share каталогында ақпараттардың бөлінетін файлдары сақталады. /Usr/indude каталогы алдымен өңдеушілерге қызықты— онда басты h-файлдар

құралады.



7.1- сурет. кәдімгі UNIX-жүйе каталогының құрылымы

/var каталогында ОЖ ағымдағы жұмыстармен байланысты файлдар сақталады. /Var/lock каталогы енді бос емес, бөлінбейтін ресурстарға қолжетімділікті алдын ала құлыптайтын файлдарды құрайды. /Var/log каталогы ОЖ жұмысы кезінде болған оқиғалардың барлық деректері жазылатын, жүйелік журналдарды сақтауға арналған. /Var/run каталогын қазіргі уақытта сол немесе басқа утилиттің қосылғанын көрсететін файлдар құрайды. /var/spool каталогы мөр басатын тапсырмалар мен пошталық хабарламалар буферін құрайды.

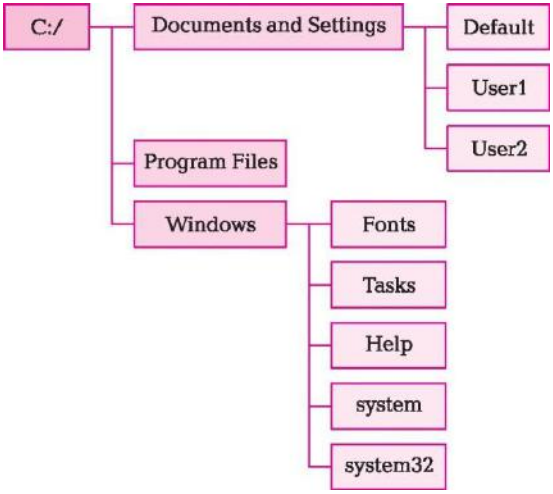
Windows XP тобының операциялық жүйелерінде каталогтар

құрылымы 7.2. - суретте көрсетілгендей түрге ие.

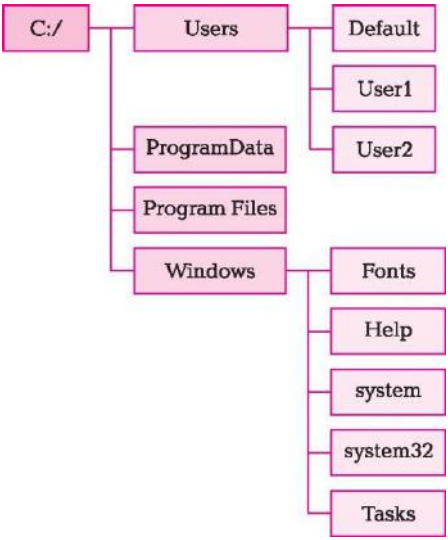
Profiles каталогында жүйеде тіркелген, әр пайдаланушыларға арналған баптаулар сақталады. Administrator папкасында жүйе әкімшісі–пайдаланушысының баптаулары сақталады, All Users папкасында жалпы барлық пайдаланушыларға арналған баптаулар сақталады. Default User папкасында жаңадан жасалатын, пайдаланушылардың барлығына қолданылатын әдеттегі баптаулар сақталады.

Program Files каталогында әдетте, жүйе пайдаланушылардың немесе әкімгерлердің орнатылған бағдарламалары сақталады.

Windows каталогында операциялық жүйенің негізгі баптаулары, сондай-ақ операциялық жүйеде орындалатын негізгі файлдар сақталады. Fonts каталогшасы, жүйеде орнатылған барлық қаріптерге, ал Help папкасы— жүйенің анықтамалық файлдарына ие. System және system 32 папкасында негізгі жүйелік утилиттер, басқару панелінің апплеттері, операциялық жүйелердің алдын ала орнатылған утилиттері (калькулятор, символдарды қарау және т.б.) және басқа утилиттер, бағдарламалар сақталады.



7.2- сурет. Windows XP кәдімгі жүйесінің каталогтар құрылымы



7.3- сурет. Windows Vista и Windows 7 кәдімгі жүйесінің каталогтар құрылымы

Windows Vista және Windows 7 тобының операциялық жүйелерінде каталогтар құрылымы Windows XP салыстырмалы түрде өзгерді (7.3-сурет).

Profiles каталогының орнына Users каталогы қолданылады. All Users папкасы жоғалып кетті, оның орнына жүктелім дискісінің түбір каталогында ProgramData папкасы пайда болды. Ал Users каталогында алдыңғы нұсқамен үйлесімділікті сақтау үшін ProgramData папкасына көрсететін All Users атты сілтеме пайда болды. Default папкасы, енді Default User папкасының орнына қолданылады, ал Users папкасындағы үйлесімділік үшін Default пакасына Default User сілтемесі жасалды. Қалған файлдық жүйе құрылымында елеулі өзгерістер болмады және өзінің бұрынғы ұйымдастырылуын сақтап қалды.

7.2. ФАЙЛДЫҢ ТҮРЛЕРІ

Файлдық жүйе әр түрлі мақсаттағы файлдарды құруы мүмкін. Файл мақсаттарын анықтау нұсқаларының бірі жоғарыда қарастырылды – орындалатын файлдардың «орындалатын» (x) қолжетімділік құқығын анықтау.

Бағдарламалар сияқты мұндай файлдарды іске асыру кезінде, операциялық жүйе процесті жасауға және бұл бағдарламаны іске асыруға әрекет етеді.

Бұдан басқа, UNIX-жүйелерінде файлдардың басқа түрлерін беруге мүмкіндік беретін, файл атрибуттары анықталады. ls -l командасының шақырылуынан кейін, файл атрибуты бірінші бағанның бірінші орына шығарылады. Осылай, төменде келтірілген үлгіде texmf және tk8.4, d атрибутына ие, бұл файлдың каталог болып табылатынын білдіреді (файлдар мен каталогтардың арасындағы айырмашылықтарды 2 – тараудан қараңыз.):

```
drwxrwx --  4  nikita users     0 Apr 28 23:25 texmf 375
-rwxr-x --  1  nikita users  Apr 26 04:27 textmode.o 452
-rwxr-x --  1  nikita users  Apr 26 04:27 readmode.o
drwxrwx --  2  nikita users     0 Apr 28 23:22 tk8.4 2948
-rwxr-x --  1  nikita users  Sep 2 20:03 tkConfig.sh
```

файлдар мен каталогтардан басқа, UNIX сүйемелденетін файлдық жүйенің көбісінде файлдардың басқа түрлері де жасала береді, мүмкін болатын барлық файлдарды дұрыс жіктеу ұсынылады.

Атап өту қажет, неше түрлі файлдарды өңдеуде ОЖ әр сүйемелденетін файлдармен жұмыс істеуге арналған жүйелік шақыртулардың әр түрлі жинағын қолданады. 2 - тарауда файлдар мен каталогтар жұмыстары үшін жүйелік шақыртулар арасындағы айырмашылықтар суреттермен сипатталды.

Қарапайым файл. Бұл файлдың түрі көп таралған. Бұл файлдарды операциялық жүйе сатылы байттар ретінде қарайды, олардың барлық құрамын өңдеу қолданбалы бағдарламамен жасалады.

Каталог. Каталогтар — бұл файлдардың орналасуының иерархиялық құрылымын берудің негізгі құралы. Каталогтың ішкі құрылысы бойынша— бұл ақпараттар блогындағы файлдарға сәйкес келетін метаақпараттарды құрайтын диск блогына сілтемелер мен файлдарды құрайтын аттары тізілген арнайы түрі.

Символдық сілтеме. Кітаптың 2 - тарауында, бірнеше аттары бар ақпараттардың бір жинағына ат қою тәсілі ретінде қатқыл сілтеме қарастырылған болатын. Әр қатқыл сілтеме деректер жинағының метаақпараттық блог сілтемесіне және деректер жинағына сәйкес келетін файл атын анықтаушы - каталог элементі болып табылады.

Қатқыл сілтемедеге қарағанда символдық сілтемеде осы сілтемені көрсететін файлдардың аты бар (толық немесе салыстырмалы жолмен) жолақ сақталған файлдың арнайы түрін ұсынады. Егер символдық сілтемеде аты сақталған файлды өшіретін болса, онда символдық сілтеме дұрыс емес болады және «бос» деп көрсетіледі.

Символдық құрылғы. Символдық құрылғылар файлының көмегімен пайдаланушы бағдарламалары операциялық жүйені сүйемелдейтін әртүрлі құрылғыларға жүгіне алады. Құрылғы файлына жазылатын деректерлер оған шынымен берілуі үшін операциялық жүйенің ядросы құрылғының осы түрін сүйемелдеуі міндетті. Егер құрылғыны операциялық жүйе ядросындағы сәйкес келетін драйвер сүйемелдемесе құрылғының файлы бәрібір дискіде бола береді. Бірақ құрылғыға деректер түспейді.

Символдық құрылғы файлы арқылы қолжетімді құрылғымен дерек алмасу, символдық режим бойынша сатылы жүреді— бір оқу немесе жазу операциясы үшін жазбалар құрылғыға беріледі немесе одан тек бір символ оқылады. Файлдар, физикалық құрылғылар ретінде (мысалы, сатылы порт /dev/ttyS0), сондай-ақ, пайдаланушы жұмысының ыңғайлылығы немесе қолданбалы бағдарламалардың алгоритімін жеңілдету үшін операциялық жүйені сүйемелдейтін логикалық түрінде ұсынылуы мүмкін. Логикалық құрылғыларға әртүрлі желілік хаттамаларды (PPP хаттамалары үшін /dev/ ppp) сүйемелдеуге арналған құрылғылар файлдары мысал бола алады.

Блогтық құрылғы. Блогтық құрылғының файлдары, белгіленген өлшемді деректер блогына жазу, оқуды бір операциясына қабылдауын талап ететін, құрылғылармен ақпарат алмасуға арналған. Мұндай құрылғыға қатқыл дискі тарауы мысал бола алады: дискіге жазылатын деректер, дискінің еселі секторына блок өлшемді болып түседі. Қалған блогтық құрылғыларының файлдары, символдық құрылғы файлдарына ұқсас болады.

Атаулы канал. Атаулы каналдардың файлдары**,** жіберілген процестер арасындағы деректердің алмасуына арналған. Атаулы канал, деректерді кез келген процеспен жазуға болатын, бірақ кейін басқа процеспен оқылатын, кезекті ұсынады. Осылайша ортақ міндетті шешетін, процестердің біріккен жұмысын қамтамасыз етуге болады. Атаулы каналға ауыстырылатын деректер көлемі, тек ерікті дискілік кеңістікпен шектеледі, деректердің құрылымы процестермен анықталады.

Доменді ұяшық. Доменді ұяшықтар, әр түрлі процестер арасындағы деректер алмасуы үшін арналған. Олардың атаулы каналдардың басты айырмашылығы доменді ұяшықтардың көмегімен (сокет деп аталатын) процестер арасындағы - желіде біріктірілген әртүрлі физикалық компьютерде орындалатын деректер алмасуды ұйымдастыруға болады. Доменді ұяшықтардың қолданумен деректерлермен алмасу үшін TCP/IP хаттамалардың жіңішке қамшысы қолданылады, бұл бір компьютердің шегінде де, сондай-ақ желіде де деректерлермен алмасуға мүмкіндік береді.

BASH тіліндегі міндеттердің орындалуы кезінде файлдың түрін анықтау үшін төменде келтірілген test командасының кілттерін қолдануға болады:

- -f <файл> — файл бар және қарапайым файл болып табылады;
- -d <файл> — файл бар және каталог болып табылады;
- -c <файл> — файл бар және символдық құрылғы болып табылады;
- -b <файл> — файл бар және блогтық құрылғы болып табылады;
- -p <файл> — файл бар және атаулы канал болып табылады.

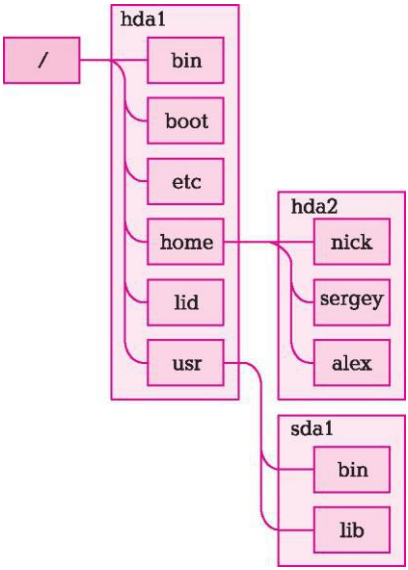
7.3. ФАЙЛДЫҚ ЖҮЙЕЛЕРДІ ҚҰРАСТЫРУ

UNIX – жүйесіндегі пайдаланушыға қолжетімді каталогтар мен файлдар, каталогтардың тұтас иерархиялық жүйесінің көмегімен құрылған. Егер бұл жүйені оның логикалық құрылымы тұрғысынан қарасақ — пайдаланушыға жинақтағыштағы файлдардың физикалық таралу мәселелері жасырын болып қалады — каталогтардың тұтас құрылымы, әр түрлі дискідегі файлдарды біріктіре алады. Сонымен қатар, физикалық бір дискіде орналасқан файлдар мен каталогтар, жалпы жүйеде ағаш тәрізді ұсынылады. Бұл дискінің физикалық түбірі каталог құрамы, жалпы жүйенің кейбір каталогтары түрінде ұсынылады. Нақты физикалық тасымалдаушыда сақталатын файлдардың таралуына қатысты каталог, *құрастыру нүктесі* деп аталады, ал дискі құрамы - операциялық жүйенің пайдаланушысына қолжетімді болатын процестің өзі — *дискіні жөндеу* деп аталады.

7.4. суретте тұтас файлдық жүйеге құрастырылған 3 дискілік жинақтағыш көрсетілген — hda1, hda2 и sda1. Сонымен қатар, құрастыру нүктесі - home каталогы және usr каталогы, түбірлі каталог болып табылады, яғни hda1 дискісі негізгі каталогтар жүйесін құрайды, home және usr каталогтарына hda2 және sda1 дискісін құрайтын каталогтар құрастырылады. физикалық дискілер әр түрлі файлдық жүйеге ие, бірақ олар бәрібір тұтас файлдық жүйеге біріктіріледі. Мысалы, UNIX-ұқсас ОС Linux үшін стандартты файлдық жүйе ретінде ext2, ал CD- дискілер үшін стандартты файлдық жүйе болып ISO 9660 қызмет етеді.

CD-жетектер және басқа да ауысатын жинақтағыштар — жұмыс процесі кезінде құрастыруды талап ететін құрылғының негізгі түрлері. CD- ROM- да орналасқан файлдарға қол жеткізу үшін оны міндетті түрде жетекке салып қана қоймай, оны құрастыру керек. Тек осыдан кейін ғана дискідегі файлдар қолдану үшін қолжетімді болады.

UNIX-жүйелерінде дискіні құрастыру үшін mount командасы қызмет етеді. Оны қолдану үшін келесі параметрлерді көрсету керек: файлдық жүйенің түрі, құрастыру нүктесі және дискілік жинақтағыштағы құрылғы файлының аты. Жинақтағыш құрылғысының файлы блогтық құрылғы түріне жатады және /dev каталогында орналасады.



7.4-сурет. Физикалық тасымалдау бойынша каталог құрылымын тарату үлгісі

Linux операциялық жүйелері (devfs орнатылған кеңейтулерінсіз) үшін құрылғыларды атау ережесі мынадай:

- IDE-жинақтаушысы hd- н; SCSI-жинақтағышы sd-н басталатын аттарға ие;
- IDE-жинақтағыштары үшін шина және құрылғының қосылуы көрсетіледі, яғни hda Primary Master дискісіне сәйкес келеді; hdb — Primary Slave, hdc — Secondary Master; hdd — Secondary Slave. SCSI - жинақтағышы үшін бақылаушыдағы құрылғының нөмірі көрсетіледі. Басқаша айтқанда, sda — бақылаушыдағы бірінші құрылғы, sdb — екінші және т.б.
- Әдетте жинақтағыш емес, ал олардың бөліктері құрастырылатын болғандықтан, бөліктер жинақтағыштың атынан кейін сан түрінде көрсетіледі. Сонымен қатар, 1 — 4 бірінші бөліктер, 5-тен әрі қарай екінші бөліктердегі логикалық дискілерді береді.

Мысалы, файл Primary Master портына қосылған IDE-құрылғысындағы екінші логикалық дискі үшін құрылғы /dev/hda6 атына ие болады. DOS немесе Windows- ке қолжетімді, тек бірінші бөліктің тек біреуіне ғана бұл атау E дискісінің әрпіне сәйкес келеді:

dev/hdc жетегінде жатқан CD-ROM дискісін құрастыру үшін келесі командаларды орындау қажет:

```
mount -t iso9660 /dev/hdc /mnt/cdrom
```

мұнда iso9660 — файлдық жүйенің түрі тип, /dev/hdc — құрылғының атауы, /mnt/cdrom — құрастыру нүктесі.

Құрылғымен жұмысты аяқтағаннан соң, оны құрастыруға, яғни файлдық жүйеден өшіруге болады. Бұл umount командасымен жүргізіледі, параметрі ретінде құрылғының файлы немесе құрастыру нүктесі көрсетіледі. Алдыңғы үлгіде құрастырғандай, CD-ROM-ды қайта құрастыру үшін келесі команданы орындау қажет:

```
umount /dev/hdc
```

немесе

```
umount /mnt/cdrom
```

операциялық жүктеу кезінде дискі бөліктерін автоматты құрастыру үшін құрастыру реті мен құрастырылатын жинақтағыштар анықталатын /etc/fstab файлы қызмет етеді.

Файл келесі параметрлерге ие:

```
/dev/hda2    /             ext2 defaults      0    1
/dev/hdc     /mnt/cdrom    auto noauto,ro     0    0
```

Файлдың бірінші бағанында – құрылғы аты, екіншіде – құрастыру нүктесі, үшінші – файлдық жүйенің түрі, төртіншісінде – құрастыру параметрлері көрсетіледі. Defaults параметрі әдеттегідей құрастыруды шақырады, noauto жүктелім кезінде файлдық жүйені құрастыруды болдырмайды, ro жинақтағышқа жазуға тыйым салады. Соңғы екі баған жүктелім кезінде жинақтағыш қабатын тексеру режимін басқарады.

Егер дискілік жинақтағыш /etc/fstab файлында көрсетілсе, онда оны құрастыру үшін mount командасының параметрі ретінде, тек құрастыру нүктесін көрсетсе болады, қалған параметрлер файлдан оқылатын болады.

БАҚЫЛАУ СҰРАҚТАРЫ

1. UNIX -ң қандай каталогтары орындалатын файлдарды сақтайды?
2. Windows XP және Vista файлдық жүйесінің құрылысы қалай ерекшеленеді?
3. Символдық және блогтық құрылғы файлдарының айырмашылықтары қандай?
4. Құрастыру нүктесі дегеніміз не?
5. Атаулы канал дегеніміз не? Ол қалай қолданылады?

8 ТАРАУ

ПАЙДАЛАНУШЫЛАРДЫ БАСҚАРУ

8.1. ПАЙДАЛАНУШЫЛАР МЕН ТОПТАРДЫҢ ЖАСАЛУЫ

UNIX - жүйелер көп пайдаланушылары ОЖ болып табылады. UNIX-жүйесі орнатылған компьютермен тек бір адам қолданатын болса, жүйемен жұмыс жасау үшін әкімшінің есеп жазбасы root – пайдаланушының есеп жазбасының біреуі болса да жеткілікті.

Әкішінің есеп жазбасын қолдана отырып, қарапайым жұмысты орындауға, қауіпсіздік есебінен ұсынылмайды — ойланбай әрекет еткен жағдайда, жүйеге айтарлықтай зиян келтіруге болады. Мұндай жағдайды болдырмас үшін әкішіге қарағанда құқығы аз есеп жазбасымен жұмыс жасауға болады.

Пайдаланушының жаңа есеп жазбасын жасау үшін useradd командасы қолданылады, мұны root пайдаланушысы орындай алады. Іске асырудың қарапайым формасында

```
useradd <логин>
```

ол /etc/passwd файлында бірінші UID және GID-мен, /home/<логин> түріндегі үй каталогымен және әдеттегідей командалық интерпретатормен (әдетте /bin/bash) жаңа тіркелім жазбасын шығарады.

UID, GID, үй каталогы мен командалық интерпретаторды нақты беру үшін команданың кең таралған түрін қолдануға болады:

```
useradd -u <UID> -g <GID> -d <каталог> -s
<интерпретатор> <логин>
```

Мысалы, vasya UID = 10001, GID = 200, /home2/vasya үй каталогы және /bin/zsh әдеттегідей командалық интерпретатормен жаңа есеп жазбасын шығару үшін келесі команданы орындау қажет : `useradd -u 10001 -g 200 -d /home2/vasya -s /bin/zsh vasya`

Есеп жазба пайда болғаннан кейін пайдаланушы жүйеге кіру құқығына ие болмайды. Бұл құқықты алу үшін тіркеу жазбасы үшін passwd <логин> командаларымен орындалатын парольді анықтау қажет. Бұл команданы шақыруға жауап ретінде, парольді енгізу және оны тексеру үшін қайта енгізу сұранысы шығады. Параметрсіз іске асырылған кезде, password командасы ағымдағы пайдаланушының паролін өзгертеді және алдымен бұрынғы парольді сұрайды.

Топтарды қосу үшін groupadd командасы қызмет етеді және ол іске асырылудың келесі параметріне ие:

```
groupadd -g <GID> <топ аты>
```

яғни, powerusers және GID = 200 атты топтарды құру үшін:

```
groupadd -g 200 powerusers -
```
орындау жеткілікті.

Қазіргі Windows операциялық жүйелері көпқолданысты болып табылады, демек онда бірден көп есеп жазбасын жасауға болады. Windows-та жасауды, қарауды немесе пайдаланушыны өшіруі үшін net user командаларының біреуін қолдануға болады:

```
net user [логин {пароль | *} /add [ключи] [/domain]]
net user [логин [пароль | *] [ключи]] [/domain] net user
[логин [/delete] [/domain]]
```

Бірінші команда - жүйеге жаңа пайдаланушыны қосуға арналған, ал екінші — бар пайдаланушының құрамын немесе парольді өзгертуге арналған, ал үшінші — бар есеп жазбасын өшіруге арналған.

Егер осы командаларды енгізерде, парольмен бірге «*» символын басса, онда командалардың орындалуы алдында, парольді енгізу өтінішімен шақыру пайда болады. Net user командасының жолағындағы аргументтердің бірі ретінде пароль енгізілген жағдайға қарағанда бұл режимде терілген пароль экранда көрсетілмейді.

Берілген операциялардың қосымша опциялары ретінде келесі кілттер қолданылуы мүмкін:

- /active:{no | yes} — көрсетілген тіркеу жазбаларын (yes) қосады; немесе (no) сөндіреді;
- /expires:{{дата} | never} — берілген есеп жазбасының белсенділік мерзімінің өтуін орнатады немесе берілген есеп жазбасының

ешқашан өшпейтінін хабарлайды.

- /homedir:{nyTb} — берілген жолды үй каталогы ретінде ұсынады және бұл жол болуы керек;
- /passwordchg:{yes | no} — пайдаланушыға өзінің есеп жазбасында парольді өзі өзгертуіне рұқсат береді (yes) немесе тыйым(no) салады;
- /passwordreq:{yes | no} — берілген есеп жазбасына парольді қолдануды (yes) талап етеді немесе есеп жазбасын парольсіз қолдануға (no) тыйым салады;
- /times:{время | all} — пайдаланушының жүйемен жұмыс істеуіне уақыт қояды.

Егер есеп жазбасының паролі өзгерсе, онда жаңадан орнатылатын парольдің минималды ұзындығы, net accounts /minpwlen командасының көмегімен орнатылған ұзындығынан кіші болмауы керек.

8.2. ПАЙДАЛАНУШЫ СЕАНСЫН ЖҮКТЕУ ФАЙЛДАРЫ

UNIX және Linux сеанс жүктеуінің басында пайдаланушымен (оның жүйеге сәтті кіруінен кейін, бірақ командалық жолдың шақыру сәтіне дейін) міндеттер орындалады, пайдаланушының бастапқы ақпараттық шеңберін жасайды. Бұл міндеттер ауыспалы шеңбердің мәндерін қояды, пайдаланушы терминал жұмысының режимін анықтай алады, дискілерді құрастырады.

Сеанс жүктелуі кезіндегі файл атауы пайдаланылатын командалық интерпретаторға байланысты, сондықтан BASH командалық интерпретаторы қолданылады деп тұжырымдаймыз.

Жүктелім сеансының екі файлы бар — жалпыжүйелік, жүйенің кез келген пайдаланушысының бастапқы сеансында орындалатын міндеттер құрайды және жүктелімнің қолданыс файлы, әр жеке файлға тән міндеттерді құрайды.

Сеанс жүктелімінің жалпы жүйелік файлының атауы /etc/ profile. Ол барлық пайдаланушылардың оқуына қолжетімді. Бұл файлдың құрамын тек жүйе әкімшісі өзгерте алады. Әдетте бұл файл терминалдың бастапқы орнатылымдарын, сондай ақ орындалатын файлдар мен динамикалық жүктелетін кітапханаларға жол беретін ауыспалы шеңберлерді анықтайды. Мұндай файлдың фрагмент үлгісі төменде келтірілген:

```
export
PATH=/bin:/sbin:/usr/bin:/usr/sbin:/usr/X11R6/bin

export     LD_LIBRARY_PATH=/lib:/usr/lib:/usr/local/lib
export PS1="$"
```

Бұл файлда LD_ LIBRARY_PATH – ң ауыспалы мәндерін орнату көмегімен, динамикалық кітапханаларды іздеу жолдары анықталады, ал ауыспалы PS1 командалық интерпретаторды шақыруын береді.

Әдетте бұл файл:

```
if [ -d ~/bin ] ; then
PATH="~/bin:${PATH}" fi

ENV=$HOME/.bashrc USERNAME="admin" export ENV USERNAME
PATH=$PATH:/usr/local/spice/bin:.
export EDITOR=le
alias ls='ls --color=auto'
```

параметрлерін жиі қолданылатын командалардың қысқаша аттарын – командаларын анықтау үшін әдеттегідей мәтіндік редакторды анықтау үшін басқа ауыспалы шеңбердің мәндерін орнату және ауыспалы PATH-та пайдаланушы бағдарламаларына жолдарды орнату үшін қызмет етеді.

Осылайша, жоғарыда келтірілген .profile файлдың фрагменті PATH ауыспалысына ~/bin каталогын қосады, мұнда пайдаланушы өзінің орындалатын файлдарын салады. Егер мұндай каталог жоқ болса, оларды қосу жүрмейді. Одан кейін ENV және USERNAME ауыспалыларының мәндері қондырылады, PATH ауыспалысына /usr/local/spice/bin жолы және ағымдағы каталог «.» қосылады, әдеттегідей мәтіндік редактордың атауы орнатылады. ls командасы үшін соңғы жол болып алиас беріледі. Оның орнатқаннан кейін ls командалық интерпретаторын енгізгенде, оны ls --color=auto командасы қабылдағандай көрсетіледі. --Color=auto параметрі каталогтар мен файлдардың атауын терминалға түсін шығару мүмкіндігі жағдайында, олардың түрлеріне байланысты әртүрлі түспен ерекшеленетінін анықтайды.

Пайдаланушыларға арналған Windows операциялық жүйелерінде сеанс жүктелімінің файлдары орнатылмайды. Бірақ жеке пайдаланушылар үшін мұндай файлдарды беретін мүмкіндік бар. Бұл үшін net user командасымен /scriptpath:<жол> кілтін қолдану керек. Мұнда жүктелім файлына жол - <жол> . Сонымен қатар, осындай жолдарға нақты шектеулер қойылады. Мысалы, бұл жол абсолютті жол емес, %systemroot%\System32\Repl\Import\Scripts каталогына қатысты жол болады. Жаңа есеп жазбасын жасауға болады және оған сеанс

жүктелімінің файлын орнату мынадай:

```
NET USER user password /ADD /HOMEDIR:\\Server_05\
/scriptpath:logon.cmd /DOMAIN
```

Сонымен қатар, \\Server 05\ үй каталогын password паролімен user пайдаланушысы жасай алады және компьютерге кіретін пайдаланушы осы доменде жасалады.

БАҚЫЛАУ СҰРАҚТАРЫ

1. Операциялық жүйелердің пайдаланушылары туралы жазбаларды басқаруда қандай негізгі міндеттер шешіледі?
2. Пайдаланушылардың топқа біріктірілуі қандай рөл атқарады?
3. Пайдаланушының жұмыс сеансы жүктелуінің командалық файлында қандай әрекеттер орындалады?
4. Windows жүйесінің жүктелім файлдарының ерекшеліктері немен тұжырымдалады?

9 ТАРАУ

UNIX ЖӘНЕ WINDOWS ҚОЛДАНБАЛЫ БАҒДАРЛАМАЛАУ

9.1.

ТАҚЫРЫПТЫҚ ФАЙЛДАР

Командалық интерпретатор тілінде тапсырмаларды жазу, пайдаланушыға қолжетімді командалардың тізімін кеңейтеді — орындалатын файл түрінде рәсімделген әр тапсырма, жіберілген команда сияқты болуы мүмкін. Мұндай командалар мүмкіндігінің шектеулі болуы, мұнда тапсырмаларды басқару тілдері, басқа дайын командаларды құрамдастыруға ғана мүмкіндік береді (орындалатын файлдар түріндегі интерпретатордың сыртқы және ішкі командалары), бірақ жүйелік шақыруларға - операциялық жүйелердің ядро функциясына ешқандай қолжетімділік мүмкіндігін ұсынбайды. Жүйелік шақыртуларды қолдану қажеттілігі кезінде жоғары деңгейдегі тілдердің (сирек —ассемблер тілінде) бірімен жазылған бағдарламалар қолданылады.

UNIX - жүйесінің бұдан әрі мазмұндалатын негізгісі болатын С тілі көп кездесетін стандартты тіл болып саналады. Сонымен қатар, UNIX нұсқаларының барлығында дерлік, стандартты жүйелік шақыртулар қолданылады. Оқырман осы С тіліндегі бағдарламалаумен тұжырымдалады, сол себепті осы тарауда UNIX-ке тән тек негізгі ерекшеліктер келтірілген — компилятордың командалық жолдары параметрлері және жүйелік шақырулар форматын анықтайтын тақырыптық файлдар ғана қарастырылған.

UNIX және Linux - /usr/ include каталогында тұратын, жүйелік шақырулардың анықтамасын құрайтын стандартты тақырыптық файлдар. Әр файл деректер түрі мен нақты функциялардан тұрады. Сонымен қатар, жүйелік шақырулардың анықтамасы үшін файлдардың стандартты түрін қолданудан бас тарту тән.

```
#include <time.h>
clock t clock(void);
```

Мысалы, жүйелік шақыру - процес басталу сәтінде өтетін микросекунд санын қайтаратын, clock() функциясын шақыратын

clock_t түрін қолданады. Сол time.h тақырыптық файлында clock_t түрі long тәрізді анықталады.

Бірақ, егер UNIX қандай да бір нұсқаларында бұл тип өзгерсе, (мысалы, long long өзгерсе), функциялардың тақырыптар үйлесімділігі сақталады.

9.1 – кестесінде UNIX және Linux жүйелерінде негізгі тақырыптық файлдардың атауы саналған, олар жүйелік шақырулар анықтамасын және кітапханалық қызметтерді, осы файлдың құрамына қатысты қысқаша түсініктемелерден тұрады. Файлдардың атауы /usr/include каталогына қатысты көрсетілген.

9.1- кесте. UNIX/Linux – те кітапханалық функциялар мен жүйелік шақыруларды құрайтын негізгі тақырыптық файлдар.

Тақырыптық файл	Тақырыптық файлдың құрамы
dirent.h	Каталогпен жұмыс жасауға арналған деректер түрлері мен функциялары— каталогтан файлдар тізімін алу үшін мета деректер файлдарына жүгіну үшін және т.б.
errno.h	Қате жағдайларды өңдеуге арналған макроанықтамалар мен тұрақты шамалар, функциялар, сандық кодты қателерді мәтіндік формаға түрлендіру.
fcntl.h	Файлдармен жұмыс жасау үшін макроанықтамалар және деректер түрлері, функциялары— оларды жасау, ашу, атрибуттарын өзгерту үшін.
float.h	Шатасқан үтірлі сандармен жұмыс жасау үшін деректер түрлері мен функциялары
limits.h	Операциялық жүйелерге шектеу қоятын тұрақты шамалар — файл атының максималды ұзындығын, деректер түрлерінің өлшемдері, аталатын жадының максималды көлемін және т.б. көрсетеді
math.h	Тригонометриялық, логрифмикалық, деңгейге енгізу, т.б. – әртүрлі математикалық операцияларды орындауға арналған функциялар
pwd.h	Парольдер файлында пайдаланушылар туралы ақпарат алу функциясы— /etc/passwd парольді файлымен жұмыс жасау үшін функциялар мен ақпарат түрлері
regex.h	Жиі өрнектермен жұмыс жасау үшін ақпарат түрлері мен функциялар. Жиі өрнектер — төселуші символдардың кеңейтілген нұсқасы (анығырақ 5- тарауды қараңыз)

Тақырыптық файлдар	Тақырыптық файлдардың мазмұны
signal.h	Сигналдарды өңдеу үшін ақпарат түрлері мен функциялары (анығырақ 10- тарауды қараңыз)
stdarg.h	Аргументтердің ауыспалы санды функцияларымен жұмыс жасау үшін ақпарат түрлері мен функциялары
stddef.h	Стандарттың түрлерін анықтау
stdio.h	Енгізу/ шығару стандартты кітапханалық ақпарат түрлері мен функциялары
stdlib.h	Стандартты кітапханалар ақпараттарының түрлері мен функциялары
string.h	Конкатенация, көшіріп алу, салыстыру және т.б.: жолдарымен жұмыс жасау үшін ақпарат түрлері мен функциялары
time.h	Уақыт кесінділерін өлшеу, жүйелік уақытты есептеу және орнату – жүйелік таймермен жұмыс жасау үшін ақпарат түрлері мен функциялары
unistd.h	Жүйелік шақырулар жұмысының әртүрлі режимін беру үшін макроанықтауыштар және негізгі жүйелік шақырулардың функциялары
sys/ipc.h	Процесаралық IPC өзара әрекетті сүйемелдеу үшін ақпарат түрлері мен функциялары (10- тарауды қараңыз)
sys/sem.h	Семафорамен жұмыс жасау үшін ақпарат түрлері мен функциялары (10-тарауды қараңыз)
sys/types.h	Ядроның кестелік жүйесімен жұмыс жасау үшін ақпарат түрлері

Windows операциялық жүйелерінде UNIX және Linux операциялық жүйелерімен салыстырғанда C тіліндегі әзірлемелер жүйесі әдеттегідей қолжетімсіз, ол қосымша орнатылымдарды талап етеді. Windows құрамының ОЖ тәрізді жүйелердің бүтін қатары бар, дербес жағдайда, C компилятор тілін қоса алатын Microsoft Visual Studio, әзірлемелік жүйесі бар. Сондай-ақ Windows үшін бейімделген, Linux-те кең таралған, мысалы MinGW немесе cygwin.gcc құрамында жүретін, компиляция жүйесінің бірнеше түрлері бар.

Windows-те компиляция жүйесінің осындай ерекшеліктерінен, стандартты тақырыптық файлдары бастапқыда қолжетімді болмайды, олар таңдалған әзірлемелік жүйелерімен бірге қондырылады.

Бұл файлдардың орналасуы, таңдалған әзірлемелік жүйенің қандай каталогқа орнатылғанына байланысты.

Бұдан басқа, Windows операциялық жүйелері үшін арнайы Windows SDK (Software Development Kit) кітапханасы бар. Бұл кітапхана Microsoft фирмасымен жасалады және өңделеді. Windows тәрізді қосымшаларды әзірлеуге қажетті тақырыптық файлдар мен жүйелердің барлық өзекті жағдайын сүйемелдеуге арналған. Осы жолдарды жазу сәтінде SDK соңғы өзекті нұсқасы - Microsoft Windows SDK for Windows 7 and .NET Framework 4 нұсқасы болды. Windows SDK нұсқасында Microsoft Visual Studio 2010 әзірлемесінің құнды ортасы жүреді.

9.2. кестесінде жүйелік шақыртулардың анықтамаларын, кітапханалық функцияларды, осы файлға қатысты қысқаша түсініктемелерден тұратын Windows операциялық жүйелеріндегі негізгі тақырыптық файлдардың атаулары келтірілген.

9.2- кесте. WinAPI в Windows шақыруларын құрайтын негізгі тақырыптық файлдар.

Тақырыптық файлдар	Файлдардың тақырыптық мазмұны
windows.h	Windows API барлық функцияларының хабарландырулары, Windows макростарының барлық ерекшеліктерін анықтауды, ақпарат түрлерін және т.б. құрайтын басты тақырыптық файл.
excpt.h	Төтенше жағдайларды өңдеуге арналған жарлықтарды құрайтын файл.
fcntl.h	Файлдармен жұмыс – оларды жасау, ашу, атрибуттарын өзгерту үшін макроанықтамалар мен ақпарат түрлері, функциялары
float.h	Шатасқан үтірлі сандармен жұмыс жасауға арналған ақпарат түрлері мен функциялары
limits.h	Файл атауының максималды ұзақтығы, ақпарат түрлерінің өлшемдері, аталатын жадының максималды көлемін – операциялық жүйелерді шектеуді беретін тұрақты шамалар
math.h	Тригонометриялық, логрифмикалық, деңгейге енгізу және т.б. - әртүрлі математикалық операцияларды орындауға арналған функциялар

Тақырыптық файлдар	Тақырыптық файлдардың мазмұны
windef.h	Әртүрлі макростар және типтері
signal.h	Сигналды өңдеуге арналған ақпарат түрлері мен функциялары
stdarg.h	Аргументтердің ауыспалы функцияларымен жұмыс жасауға арналған ақпарат түрлері мен функциялары
stddef.h	Стандартты типтерді анықтау
stdio.h	Енгізу/ шығару стандартты кітапханасының ақпарат түрлері мен функциялары
stdlib.h	Стандартты кітапхананың ақпараттар түрлері мен функциялары
string.h	Конкатенация, көшіріп алу, салыстыру және т.б.: жолдарымен жұмыс жасау үшін ақпарат түрлері мен функциялары
time.h	Уақыт кесінділерін өлшеу, жүйелік уақытты есептеу және орнату – жүйелік таймермен жұмыс жасау үшін ақпарат түрлері мен функциялары
WinSock.h	Сокеттермен жұмыс жасау үшін ақпарат түрлері және функциялары

9.2. UNIX БАҒДАРЛАМАЛАРДЫ КОМПИЛЯЦИЯЛАУ

UNIX-та бағдарламаларды компиляциялау екі кезеңде жүргізіледі. Шығыс мәтіндерінің бірінші кезеңінде cc (немесе gcc) компиляторының көмегімен объективті файлдар (.o кеңейтілуімен) қалыптасады. Екінші кезеңде ld жинаушының көмегімен орындалатын файл немесе кітапхана файлы қалыптасады.

Жинау мен компиляция процесін басқару (мысалы, кодты оңтайландыру деңгейі, шығу файлының параметрі, жүйелік кітапханаларға жолдар) жинаушы мен компиляторды іске қосу кілттерімен беріледі.

Қарапайым бағдарламаларды жинау және компиляциялау үшін бұдан әрі автоматты түрде ld жинаушысын қосатын, gcc компиляторын шақыру жеткілікті.

Мысалы, myprogram.c файлынан шығыс мәтіндік бағдарламалар негізінде myprogram орындалушы файлын алу үшін

`gcc -o myprogram myprogram.c` командасын орындау жеткілікті.

Мұнда o кілтінен кейін шығатын орындалушы файлдың атауы көрсетіледі. Кілттерден кейін құрастырушы файлдың атауы көрсетіледі. Ақпараттар типтері мен функциялардың анықтамасын құрайтын бірнеше файлдарды көрсету мүмкіндігі:

```
gcc -o myprogram myprograml.c myprogram2.c
```

Шығыстық мәтіндер файлдарын құрастыру кезінде функциялар, жобалау ауыспалылар немесе бір атты атаулармен ақпарат түрлерінің анықтамалары қатыспауы керек, мысалы main() функциясын бірнеше рет жариялау қолжетімсіз.

Тақырыптық файлдар мен кітапханалар файлдары орналасқан жолдарды көрсету үшін сәйкесінше -I және –L кілттері қолданылады. Мысалы, келесі параметрді құрастыру кезінде, компилятор /usr/local/indude каталогындағы тақырыптық файлдарды, /usr/local/lib және /usr/share/lib: каталогында – кітапханаларды іздейтін болады.

```
gcc -o myprogram -I/usr/local/include\ -L/usr/local/lib
-L/usr/share/lib myprogram.c
```

Статикалық кітапханалардың файлдары кеңейтулерге ие, ал олардың атаулары lib сөзден басталады. Мысалы, libm.a — кітапхананың аты, math.h. Тақырыптық файлдарында анықталған, математикалық бағдарламалық кодтан тұрады.

Орындалатын файлдың жинау процесі кезінде кітапханаларды қосу үшін -l кілті қолданылады. Қосылатын кітапханалардың атауы lib префиксіз және .a кеңейтуінсіз көрсетіледі. Мысалы, libm.a кітапханаларды пайдаланушы бағдарламаларды құрастыру және компиляциялау үшін

```
gcc -o myprogram -L/usr/lib -lm myprogram.c
```

шақыртуларды қолдану қажет.

Мұнда L-кілті кітапханалардың орналасуын береді, ал -l — оның атауы.

Статикалықтан басқа, UNIX-жүйелерінің көпшілігі тиелетін динамикалық кітапханаларды көтереді. Мұндай кітапханаларда сақталатын функцияның бағдарламалық кодтары, бағдарламаларды орындау уақытында динамикалық тиеледі. Динамикалық тиелетін кітапханалардың файлдары .so кеңейтулеріне ие болады, файлдардың атауы lib префиксімен басталады. Сонымен, математикалық функциялар кітапханаларының динамикалық тиелетін нұсқалары libm.so атауына ие болады.

Орындалатын файлдарды құрастыру кезінде динамикалық тиелетін кітапханалардан функциясын шақырту нүктелерін орнату үшін -

dynamic компиляторының кілтін көрсету қажет. Келесі команда, бағдарламаның шығысты мәтінді c.myprogram файлында, орындалатын файл myprogram құрастыру және компиляциялауды орындайды. Сонымен қатар, математикалық кітапхананың динамикалық нұсқасы қолданылады, ал кітапханалық файлдарды іздеу мына каталогтан жүреді: /usr/local/lib:

```
gcc -dynamic -o myprogram -L/usr/local/lib\ myprogram.c
```

Динамикалық кітапханаларды қолдану статикалықтарға қарағанда артықшылықтары бар — кітапханалық функциялардың бағдарламалық коды орындалушы файлдан тыс орналасады және бірнеше бағдарламалармен бірге қолданылады. Статикалық кітапханаларды қолдану кезінде, бағдарламалық код орындалатын файлда сақталатын болғандықтан, динамикалық кітапханалардың артығынан құтылу есебінен, дискілік кеңістікті үнемдеуге мүмкіндік береді. Дегенмен динамикалық кітапханалардың артықшылықтарының кері жақтары да бар. Динамикалық кітапханалар, олардың бағдарламаларын пайдаланушылардан бөлек, жүйеге қондырылады, қажетті кітапханалар болмаған жағдайда немесе динамикалық кітапханаларды пайдаланушы бағдарламалардың кітапханалар сәйкес келмеуі кезінде жұмысқа жарамсыз болып қалуы мүмкін.

UNIX-жүйелерінің көбісінде бағдарламалардың жұмысқа қабілетсіздігін ldd командасы анықтайды. Орындалатын файлдың атауымен осы бағдарламаның іске қосылуынан кейін, экран бетіне бұл файлмен орындалатын динамикалық кітапханалардың тізімі, кітапхана файлдарына жол және кітапханалардың болмауы жөнінде ақпарат шығады:

```
$ ldd 'which scat' libnsl.so.1 => No library
found libncurses.so.4 =>
/usr/lib/libncurses.so.4 (0x40035000)
libc.so.6 => /lib/libc.so.6 (0x40077000)
/lib/ld-linux.so.2 => /lib/ld-linux.so.2
(0x40000000)
```

Кітапханалардың болмауы туралы ақпарат – оның орнатуға тікелей көрсетеді. Кітапханалардың шиеленісуі жанамалы түрде атауы бойынша және кітапханалар файлына жолы бойынша анықталады. Мысалы, бағдарлама кітапхананы басқа каталогта іздеп, қателесуі мүмкін. Бұл жағдайда міндетті түрде кітапхананы қажетті каталогқа орнын ауыстыруы керек немесе LD_LIBRARY_PATH ауыспалы шеңберінде кітапханаларды каталог бойынша сатылы іздеуді орнату керек. Бұл ауыспалыдағы каталог тізімінің параметрі, PATH ауыспалы тізімінің параметріне ұқсайды.

Ауыспалы шеңберлер командалық жолдың компиляторымен жұмысын жеңілдету үшін де қолданылуы мүмкін. Осылайша, әр түрлі каталогтарда (мысалы, кейінге қалған ақпаратты құрайтын немесе құрамайтын кітапханалар) орналасқан бірдей атаулы кітапханалардың жинағын пайдаланумен, бағдарламалардың компиляциялау қажеттілігі туындаса онда кітапханаларға жолды ауыспалы шеңберге (мысалы, MY_LIB) ауыстыруға болады және ағымдағы қажеттілікке байланысты шеңберлердің ауыспалыларының мәндерін ауыстыруға болады. Компиляторды шақыру осылай көрініс табады:

```
gcc -dynamic -o myprogram -L${MY_LIB} myprogram.c
```

Жоғарыда қарастырылған барлық бағдарлама құрастыру және компиляциялау тәсілдері пайдаланушыдан құрастыру кезеңдерін жасырады — обьективті файлдар орындалатын файлдың пайда болуынан кейін бірден өшіріледі және пайдаланушы обьективті файлды көрместен, орындалатын файлды қабылдап отырды.

Кейбір жағдайларда екі кезең— обьективті файлдарды сақтай отырып, құрастыру мен құрастырмалау орындалады. Мысалы, кейбір кітапханалар шығыстық мәтінсіз объективті файл түрінде ұсынылады. Мұндай кітапханаларды пайдаланушы бағдарламаларды құрастыру үшін алдымен өз бағдарламасының объективті файлын алу қажет, содан кейін барлық (өзінің және кітапханалық) объективті файлдарды орындалатын файлға жинау керек.

Объективті файлды алу үшін компилятордың -c кілті қолданылады. Осылайша, команда

```
gcc -c myfile1.c myfile2.c -I/usr/local/include
```

файлдар компиляциясын myfile1.c и myfile2.c шақырады және олардың нәтижелері myfile1.o и myfile2.o обьективті файлдарында сақталады. Компиляцияға қажетті тақырыптық файлдарды іздеу - /usr/local/include каталогында жүреді.

Обьективтіге жиналған орындалатын файлдарды алу үшін оған обьективті файлдарды параметр ретінде жіберу арқылы, компиляторды шақыру жеткілікті. Компилятор файлдар бағдарламаның шығыстық мәтіні еместігін таниды, және оны құрастырушыға өңдеуге жібереді. Осылайша, команда

```
gcc -shared -o myprogram -L/usr/local/lib -lm myfile1.o
myfile2.o
```

myfile 1 .o и myfile2.o. обьективті файлдарының ішінен myprogram бағдарламасының жинағын шақырады. Сонымен қатар орындалатын файлда libm.so динамикалық кітапханалық функцияларының шақыру

нүктелерін қояды, құрастыру кезеңінде іздеу /usr/local/lib каталогында жүреді (бағдарламаның орындалуы кезінде кітапхананың орналасуы LD_LIBRARY_PATH ауыспалы шеңберінде берілетін болады).

BASH тілінде беру қиын болатын, қарапайым бағдарламаға мысал келтіреміз. Бұл бағдарлама, 100-ге көбейтілген берілген параметрі ретінде бұрыштың синус мәніне тең толық санды қайтарады. BASH –та тригонометриялық функциялар құрмаған, бірақ C- да бұл бағдарлама салыстырмалы түрде қарапайым жүреді:

```
#include <stdio.h>
#include <math.h>
int main(int argc, char **argv)
{
double res; int angle;

if (argc <= 1) return 0;

atoi(argv[1],  angle);
res  =  sin(angle)*100;
return (int)res;
}
```

Бұл бағдарламаны құрастыру үшін келесі шақыртулар жолын gcc пайдалануға болады:

```
gcc -lm -o sin -L/usr/lib -I/usr/include sin.c
```

мұндай бағдарламаны пайдаланатын міндеттер, мынадай бейнеленеді:

```
#!/bin/bash
if [ -z $1 ]; then
echo "No parameters specified"
exit 1
fi

if [ ! -z $2 ] ; then
echo "More than one parameter specified" exit 2 fi ./sin
$1
echo "Hundredths of sine of angle $1 equals " $?
```

120 параметрімен іске асырылған, мұндай тапсырмаларды экранға шығару осылай көрінеді:

```
Hundredths of sine of angle 120 equals 86
```

9.3.

WINDOWS БАҒДАРЛАМАЛАРДЫ КОМПИЛЯЦИЯЛАУ

Microsoft Visual Studio 2010 әзірлеме ортасының қолданылуымен, компиляциялауды қарастырамыз. Бұл әзірлеме ортасы тек бастаушы емес, сондай-ақ тәжірибелі бағдарламаушылардың жұмысын жеңілдететін, дамыған IDE–ны қосады. Әдетте өңдеушілерге қосымша баптауды ойламаса да болады, ол үшін ортаның автоматты баптауын қолдану жеткілікті. Дегенмен, бағдарламаларды компиляциялау процесінің жүруі туралы жалпы көріністі білу қажет, себебі кейде баптауларды әдепкі түрде өзгертуге тура келеді.

Windows-те де бағдарламаларды құрастыру екі кезеңнен тұрады. Бірінші кезеңде шығыс мәтінінен cl.exe құрастырғыштың көмегімен обьективті файл (.obj кеңейтілуімен) қалыптасады. Екінші кезеңде link.exe құрастырушысының көмегімен орындалатын файл немесе кітапханалық файл қалыптасады.

UNIX құрастырғышындағы сияқты құрастырмалау жүрісін арнайы компилятордың кілтімен басқаруға алады. Мысалы, myprogram.c файлынан бағдарламаның шығыс мәтіні негізінде, myprogram.exe орындалушы файлын алу үшін келесі команданы орындау жеткілікті:

```
cl.exe myprogram.c
```

берілген команданың көмегімен бағдарлама автоматты компиляцияланады және обьектінің файлы генерацияланады, одан кейін автоматты түрде линкер шақырылады және алынған обьектілік файлдың және стандартты кітапханалар негізінде орындалатын файл генерацияланады.

Деректерлердің функциялары мен түрлерінің анықтамасы бар бірнеше файлдарды нұсқау мүмкіндігі бар:

```
cl.exe myprogram1.c myprogram2.c
```

Мұндай шақыру кезінде, алдымен екі объективті файл компиляцияланады, одан кейін линкор шақырылады және орындалатын файл генерацияланады. Әдеттегідей орындалатын файлдың атауы, компиляциялау үшін жіберілген алғашқы файлдың атауымен анықталады. Берілген мысалда бағдарламаның аты

myprogram1.exe болады.

Тақырыптық файлдардың орналасқанын іздеу үшін /I кілті қолданылады. Мысалы, компиляциялау кезінде, келесі параметрлер компилятор ..\include каталогтан тақырыптық файлдардан іздейтін болады:

```
cl.exe /I"..\include" myprogram.c
```

Генерацияланушы шығатын файлдың атауын өзгерту үшін осы файлды қайта анықтауға мүмкіндік беретін арнайы /OUT: кілтін линкерге өткізген жөн. Сонымен қатар, опцияның компиляторға емес, линкерге жіберілетіні туралы компиляторға хабарлау керек. Ол үшін /link кілтін қолдану керек, мысалы:

```
cl.exe myprogram1.c myprogram2.c /link /OUT:myprog.exe
```

Компилятордың өзінің кілтімен де қолдануға болады:

```
cl.exe /Femyprog.exe myprogram1.c myprogram2.c
```

Бірақ әдетте мұндай ресімдер үшін компиляциялау процесі екі қадамға бөлу — жеке компиляциялау және біріктіру үшін жасалады. Компиляторға бұйрық шығару үшін жай шығыс файлын біріктірусіз компиляциялау /c опциясымен қолданылады.

```
cl.exe /c myprogram1.c myprogram2.c
```

Осы командадан кейін екі файл генерацияланады— myprogram1.obj және myprogram2.obj. Бұл файлдарды біріктіру және орындалатын бағдарламаларды алу үшін келесі команданы беру қажет:

```
link.exe /OUT:myprog.exe myprogram1.obj myprogram2.obj
somelib.lib
```

Нәтижесінде, екі обьекті файлдан— myprogram1.obj және myprogram2.obj — және бір кітапханадан — somelib.lib жиналған myprog.exe бағдарламасы шығады. Статикалық кітапханалардың файлдары .lib кеңейтулеріне ие болады.

Кітапханалардың файлдарының орналасқан жолдарын көрсету үшін /LIBPATH: кілті қолданылады. Мысалы, жоғарыда көрсетілген мысалдағы бағдарламаны жинап, бірақ кітапханалардың іздеуде қосымша каталогты беру үшін келесі команданы шақыру қажет:

```
link.exe /OUT:myprog.exe /LIBPATH:"..\libs"
myprogram1.obj myprogram2.obj somelib.lib
```

BASH сценарийін және C бағдарламасын біріктірілуін жоғарыда келтірілген мысалдарда қарастырайық. Ол үшін тек Microsoft Visual Studio және cmd.exe: командалық интерфейсін қолданамыз.

```
#include <stdio.h>
#include <math.h>
int main(int argc, char **argv)
{
double res; int angle;

if (argc <= 1) return 0;

atoi(argv[1],  angle);
res  =  sin(angle)*100;
return (int)res;
}
```

Бұл бағдарламаны компиляциялау үшін келесі cl.exe шақырту жолын пайдалануға болады:

```
cl.exe /Fesin.exe sin.c
```

мұндай бағдарламаны қолданатын тапсырма, мынадай бейнеленеді:

```
@echo off
rem Синустың есептелуі if "%1"
== "" ( echo No parameters
specified exit 1 )

if not "%2" == "" (
echo More than one parameter specified exit
2 )

.\sin %1
echo Hundredths of sine of angle %1 equals
%ERRORLEVEL%
```

бұл командалық файл жұмыстың нәтижелері Linux мысалына ұқсас.

Б
А
ҚЫЛАУ СҰРАҚТАРЫ

1. Операциялық жүйенің қандай негізгі құрауыштары қолданбалы бағдарламаның қамтамасыз ету әзірлемесінің ортасы құрамына кіреді?
2. UNIX операциялық жүйесінде C бағдарламалау тілі қандай рөл атқарады?
3. Жоғары деңгей тілінде жазылған орындалатын модульдің дайындығы кезінде қандай әрекеттер жүргізіледі?
4. Windows жүйесінде орындалушы бағдарламалық файлдың дайындығының ерекшелігі?

10-ТАРАУ

ПРОЦЕССАРАЛЫҚ ӨЗАРА ӘРЕКЕТТЕСУЛЕР

10.1.

ПРОЦЕССАРАЛЫҚ ӨЗАРА ӘРЕКЕТТЕСУДІҢ ТҮРЛЕРІ

Қазіргі кезде күрделі бағдарламалардың мағыналы бөлігінде, процессаралық өзара әрекеттесудің осындай немесе басқа түрлері де кездеседі. Бұл келесі сатылы кезеңдерден тұратын, бағдарламалық жобалау мен табиғи эволюция тәсілдерімен дәйектеледі:

1) Өзінің кодында, өзінің жұмысы үшін барлық қажетті нұсқаулықтардан тұратын монолитті бағдарламалар. Мұндай бағдарламалардың ішіндегі ақпарат алмасу функциялардың параметрлерін беру және жаһандық айнымалыларды қолдану көмегімен жүреді. Мұндай бағдарламаларды іске қосу үшін барлық қажетті жұмысты орындайтын бір процесс пайда болады;

2) Нақты анықталған шақыру интерфейсті жеке бағдарламалық модульдерді құрайтын модульдік бағдарламалар. Модульдерді бағдарламаға біріктіру, орындалатын файлдың құрастырылуы (статикалық құрастыру немесе *static linking*)кезінде немесе бағдарламаның орындалуы (динамикалық құрастыру немесе *dynamic linking)* кезінде іске асады. Модульдік бағдарламалардың артықшылығы кейбір қолданбалы кезең жетістіктерімен тұжырымдалады: бір модуль басқасымен алмасуы мүмкін. Дегенмен модульдік бағдарлама бәрібір бір процесті ғана ұсына алады, міндеттерді шешу үшін ақпараттар процестің ішінде функция параметрлері ретінде жіберіледі;

3) Процесаралық өзара әрекеттесуді қолданатын бағдарламалар. Мұндай бағдарламалар жалпы міндетті шешу үшін бағдарламалар кешенін жасап шығарады. Әр іске асырылған бағдарламалар бір немесе бірнеше процесті құрайды. Міндеттерді шешуге арналған процестердің әрқайсысы өзінің жеке ақпараттарын пайдаланады және тек өзінің жұмысының нәтижелерімен басқа процестермен алмасады немесе әртүрлі процес арасында бөлінетін, жалпы ақпараттар аумағымен

жұмыс жасайды. Процестің ерекше күрделі міндеттерін шешу үшін әр түрлі физикалық компьютерлерде іске асады және желі арқылы өзара әрекеттеседі. Процессаралық өзара әрекеттесу көп қолданбалығымен тұжырымдалады, яғни, өзара әрекеттесуші процестер, өзара әрекеттесуші интерфейстерді сақтау кезінде бір-біріне қатысусыз алмаса алады. Басқа артықшылығы, есептелетін жүктелім процесс арасында таралады. Бұл операциялық жүйеге, бағдарламалық кешеннің жеке бөліктерінің орындалуы кезінде басымдылықтарын басқаруға мүмкіндік береді, ресурс сыйымды процестердің ресурс сандарының аздығы немесе көптігін ерекшелейді.

Ортақ міндетті шешетін процестерді іске асыру үшін операциялық жүйе олардың арасындағы өзара әрекеттесуші құралдарымен қамтамасыз етілуі тиіс. Өзара әрекеттесу құралдарын ұсыну — қолданбалы бағдарламалардың емес операциялық жүйелердің міндеті, себебі алмасу механизмінің өзіне қолданбалы бағдарламаның әсерін болдырмау қажет, одан басқа алмасу механизімін сүйемелдеу пайдаланушының қарапайым процесіне қолжетімсіз, ресурстарға қолжетімділікті талап етуі мүмкін.

Процесс арасындағы өзара әрекеттесу типтік механизмдері келесі міндеттерді шешу үшін арналған:

- Бір процесстен екіншісіне ақпараттарды жіберуі;
- Бірнеше процесстердің бір деректерді бірігіп пайдалануы;
- Процесстер күйінің өзгеруін хабарлау.

10.2. ПРОЦЕССАРАЛЫҚ ӨЗАРА ӘРЕКЕТТЕСУДІҢ МЕХАНИЗМДЕРІ

Қазіргі операциялық жүйелер процесаралық өзара әрекеттесулердің мынадай жеті негізгі механизмдерін іске асырады.

Үзілістер. Бастапқыда үзілістер механизмі операциялық жүйелерде хабарлау үшін қолданылды. Нақты уақыттарда аппаратты құрылғылар кейбір жағдайлардың болуын хабарлайды (әрекеттерге дайынды, істен шығуы, ақпараттар блогының жіберілімінің аяқталуы). Мұндай жағдайлардың нұсқалары саны жеткілікті көп болуы мүмкін және олардың барлығы операциялық жүйелермен бөлінуі қажет. Дайын болғандығы туралы осындай хабарлаулар үзілістер деген атауға ие болды, себебі үзілісті қабылдау кезінде, операциялық жүйе ағымдағы міндеттердің орындалуының тоқтатуы және келіп түскен үзілістерге орай әрекет етуі керек.

Үзілістерге орай әрекет ету, әдетте, операциялық жүйеге

бағытталған, жадыда орналасқан бағдарламалық кодтың орындалуымен тұжырымдалады. Операциялық жүйе жадының арнайы аумағын (үзілістер кестесі) сүйемелдейді, мұнда әр үзілістерге (әдетте, сәйкестендірілген нөмірі бойынша) бағдарламалық код (үзілістер өңдеушісі) орналасқан жады бағытының сәйкестігі қойылады. Жүйе сүйемелдейтін үзілістер саны белгіленген. Операциялық жүйе әдетте 16-дан 256 үзілістерге дейін сүйемелдейді.

Үзілістердің өңдеушісі орындалатын процестерге тәуелді емес, дегенмен бағдарламалық код процестердің бірімен қайта анықталады. Осылайша үзілістер өңдеушісінің пайдаланушысы орнатылады. Өңдеуші орындалғаннан кейін операциялық жүйе басқаруды қайтарады немесе үзілістердің келіп түсуіне дейін белсенді болған бір немесе бірнеше міндеттердің орындалуын аяқтайды.

Құрылғыдан келіп түсетін аппаратты үзілістерден басқа, кез келген процеспен басталатын бағдарламалық үзілістер бар. Осылайша, процесс операциялық жүйеге, оның орындалуы барысында қандай да бір оқиғаның орын алғанын хабарлайды.

ОЖ жүйелік шақыруының әрінде дәл осылай үзілістер белсенді болады. Мысалы, сәйкес келуші үзілістер, мәтінді терминалдың экранына шығару кезінде белсенді етіледі.

Сигналдар. Сигналдар механизмі бағдарламалық үзілістер механизмімен ортақ қасиеттерге ие. Ол да процестердің кейбір оқиғалар туралы хабарлай алуына арналған. Сигналдардың үзілістерден басты айырмашылығы, сигналдардың көмегімен операциялық жүйені емес, бір процесс екінші процеске хабарлайды. Үзілістерге қарағанда сигналдар тағайындалған нүктеге — қабылдаушы-процесіне ие болуы керек.

Қабылдаушы-процесс сигналды қабылдауға жауап ретінде, өзінің орындалуын тоқтатады және сигналды өңдеуші- бағдарламалық кодын орындауды бастайды. Өңдеушінің коды орындалуы аяқталғаннан соң, процесс өзінің орындалуын жалғастырады. Бұл жерде үзілістерден басты айырмашылығы, әр процесс өзінің сигналдар өңдеуші жинағына ие болуы керек, ал өңдеушінің бағдарламалық кодында орналасқан жады – процестің жады болып табылады. Қарапайым сигналдар өңдеушісі — бағдарлама денесінде анықталған функция. Сигналдарды қабылдау кезінде, осы функцияның шақырылуы орын алады.

Операциялық жүйе әр процесс үшін сигналдар өңдеуші кестесін өткереді. Онда әр сигналға сигнал өңдеушісінің мекенжай сәйкестігі қойылады. Мысалы, UNIX- тәрізді операциялық жүйелер 16 және одан көп түрлі сигналдарды сүйемелдейді. Сигналдар механизмі туралы нақтырақ әрі қарай қарастырылатын болады.

Хабарламалар. Хабарлама механизмі процестер арасында ақпарат

алмасуға арналған. Ол үшін операциялық жүйе сүйемелдейтін хабарламалардың арнайы тізімі құрылады. Тізімде, процеспен жазылатын ақпараттар жинақталады. Кезекте жинақталған ақпараттар басқа процеспен «оқылуы» мүмкін, осылайша бір процестен басқа процеске ақпараттар жіберу орын алады.

Операциялық жүйе кезекте сақталатын операциялық жадының арнайы саласын сүйемелдейді. Әдетте UNIX- тәрізді операциялық жүйе 1024 дейін хабарламалар кезегін сүйемелдейді. Хабарламаны қолданатын бірінші процесс, кезекті құру қажет, ал қалған процестер құрылған кезекке қолжеткізе алу қажет. Сигнал қабылданғаннан кейін басталатын өңдеушінің орындалуында, сигналдар көмегімен өзара әрекеттесудің айырмашылығы, кезектен ақпараттарды оқу, функция көмегімен қабылдаушы-процесс жүргізеді. Жөнелтуші процесс кезектің толып қалмауын қадағалау қажет (мысалы, қабылдаушы-процесс көптен бері ақпараттарды оқымаса), себебі бұл жіберілетін ақпараттардың жоғалуына әкелуі мүмкін.

Аталған каналдар. Аталған каналдар да процесс арасында ақпарат алмасуға арналған. Дегенмен кезек ретінде оперативті жады саласы емес файлдың арнайы түрі қолданылады. Процестер бұл файлға ақпараттарды жаза алады және оқи алады. Сонымен қатар, операциялық жүйе, ақпарат алмасу үшін қолданатын барлық процестерге атаулы каналдың тең құқылы қолжетімділігін береді.

Ұяшықтар (сокеттер). Сокеттер арқылы процес аралық өзара әрекетесудің механизмдері, алдыңғы кезекте әртүрлі компьютермен орындалатын процестердің өзара әрекеттесуін қамтамасыз етеді. Әр процесс ақпарат жаза алатын және оқи алатын ұяшықтар құрады. Ұяшықтар бір-бірімен желі хаттамалары арқылы байланысады.

Процестер бір-бірімен өзара әрекеттесуі кезінде байланысқан ұяшықтар арқылы ақпарат алмастырады, ал жүйе ядросы процестер ақпаратын желі арқылы береді, әдетте, TCP/ IP хаттамалар қамшысын және желілік адаптер драйверін қолданады.

Жалпы жады. Жалпы жадтың механизмі адрестік кеңістік процесттерінде операциялық жүйеге бағытталған бір физикалық жадын көрсетуден тұрады. Жалпы жадының осы бөлігін пайдалана отырып, операциялық жүйенің ядросынсыз ақпараттармен алмасуға болады, бұл жұмыс жылдамдығын әлдеқайда арттырады.

Семафор. Жалпы жадын қолдану кезіндегі негізгі проблемалардың бірі бір уақытта қолжетімді шиеленіскен жағдайлардың алдын-алу болып табылады. Мұндай жағдай бірнеше процеспен, жадының бір бөлігіне, бір уақытта ақпарат жазу кезінде орын алуы мүмкін. Бұл жағдайда жады, процесте соңғы жазылған ақпаратты құрайды, қалғандары жоғалады. Басқа шиеленіскен жағдай, басқа процеспен жетілдірілген жады саласынан ақпараттарды бір процеспен оқуы

кезінде пайда болады. Бұл жағдайда оқылған ақпараттар көнелеу де, ішінара жаңартылған деректерді де құрауы мүмкін.

Шиеленіскен жағдайлардың алдын алу үшін семафорлар қолданылады —жадының қандай да бір бөлігін қолдану мүмкіндігін көрсететін арнайы жалаушалар. Жадыға жазу кезінде процесс семафор қойылады — бұл жадының «бос емес» екендігін білдіреді; ал жазу аяқталғаннан соң оны алып тастайды, бұл жадының босатылғанын білдіреді. Басқа процестер ақпараттарды жазу немесе оқудың алдында семафордың жағдайын тексереді және жазу мүмкін емес жағдай арнайы өңделеді. Мұндай жағдайда, процес өзінің орындалуын семафорды алғанға дейін тоқтата тұрады немесе ақпараттарды өзінің ішкі буферінде жинайды және семафорды алғанда буферден ақпараттарды жазып алады.

Семафор механизмі тек жалпы жады пайдаланғанда ғана ыңғайлы емес, кез келген біріккен ресурстардың қолданысында пайдаланылуы мүмкін.

10.3. СИГНАЛДАР

10.3.1. Жалпы түсінік

UNIX - жүйелерінің көбісі сүйемелдейтін процесс аралық өзара әрекеттесушінің көп таралған механизмі — сигналдар арқылы өзара әрекеттесу. Процесс аралық (пайдаланушының немесе жүйелік) сигналдардың алмасуы оларға орындаушы процесс үшін маңызды басқа оқиғалардың орын алуы туралы ақпараттармен алмасуға мүмкіндік береді.

Процеске сигналды беруге ықпал етуші операциялық жүйенің ядросы (мысалы, 0-ге бөлінген жағдайда ядро процеске сигнал береді, ядро төтенше жағдайларды хабарлайды), пайдаланушы (мысалы, үзілістер кезінде, [Ctrl+C] батырмасымен процес орындалуы кезінде, бұл комбинацияға сәйкес келетін процеске сигнал беріледі) немесе басқа процесс болып табылады.

Жалпы кез келген процесс онымен параллелді орындалатын басқа процеске сигнал бере алады. Сигналды беру үшін жіберуші-процесс екі параметрді анықтау қажет: PID қабылдаушы-процесс сигналы және жіберілетін сигналдың нөмірі. UNIX-жүйелері сигналдар түрлерінде шектеулі санды сүйемелдейді, әдетте 30 шамасында. 10.1- кестеде негізгі сигнал түрлерінің қысқаша сипаттамасы берілген.

Әр сигнал түрі, тіркеу нөмірі және SIGALRM түрінде мнемоникалық мәндерін иеленеді, SIG — сигналдар мнемонигінің жалпы мәні; ALRM — берілген сигнал түрінің жіберілуіне байланысты пайда болған оқиғалардың мәні.

Қабылдаушы-процеске сигналдарды жеткізу операциялық жүйемен орындалады. Қабылдаушы-процесс қабылдауға жауап ретінде, сигналды елемеуіне болады немесе берілген сигнал түрінің қабылдануымен байланысты бағдарламалық кодты орындауды бастауына болады. Код, нақты сигнал түрінің қабылдануы кезінде шақырылатын жекеленген функция ретінде рәсімделеді. Мұндай функциялар «сигналдарды өңдеуші-функциялар» атауына ие болды (немесе жай «сигналдарды өңдеуші»).

Әр процесс үшін сигналдарды өңдеушінің жиынтығы, сигналдар өңдеушінің кестесін құрады. Кесте әр процесс үшін дара. Өңдеуші кестесі жолдарының саны, операциялық жүйе сүйемелдейтін сигналдар санына тең. Кестенің әр жолында өңдеуші-функцияның бағдарламалық коды басталатын жадының мекенжайы, сигнал нөмірі жазылады — өңдеуші-функциясын көрсетуші мына формат декларациясына ие:

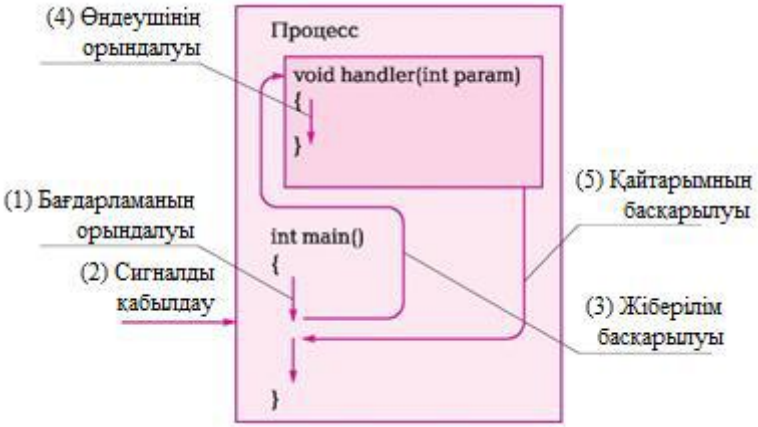
```
void handler(int sig_num);
```

сигналды қабылдау кезінде процесс өзінің қалыпты орындалуын тоқтатады және өңдеуші-функцияның бағдарламалық кодын орындауды бастайды. Өңдеуші-функция орындалуын аяқтаған соң, сигнал қабылдаған кезде орындалған команданың ізіне басқару командасымен қайтарылады (10.1-сурет).

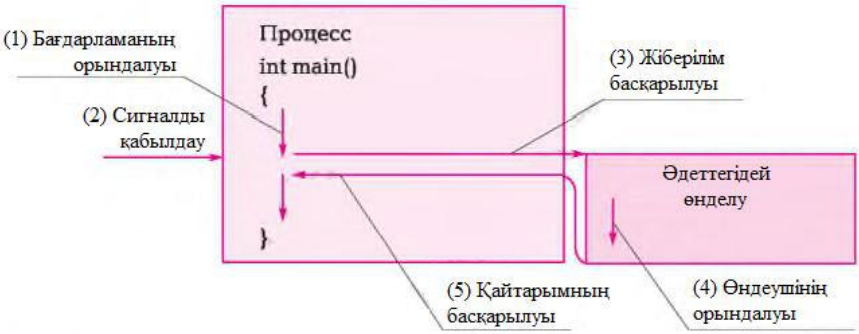
Сигналдарды өңдеуші кестенің кейбірі үшін өңдеуші-функциялар мекенжайын құрмауы мүмкін.

10.1. кесте UNIX негізгі кестелері

Сигнал нөмірі	Сигналдың мнемоникасы	Сигналдың сипаттамасы	Әдепкі әрекеттері
1	SIGHUP	Ағымдағы терминалдың байланысын үзу (фондық процесс режиміне аудару)	Процестің аяқталуы
2	SIGINT	Процестің үзілістері (әдетте [Ctrl+C] түймесімен генерацияланады)	Бірдей
3	SIGQUIT	Процестен шығу (әдетте [Ctrl+\] түймесінің үйлесімділігімен генерацияланады)	процестің аяқталуы және core файлының құрылуы (шығу сәтінде процесс жадының ахуалын құрайды)
6	SIGABRT	Процестің апаттық аяқталуы; abort () функциясымен генерациялануы мүмкін.	процестің аяқталуы және core файлының құрылуы
9	SIGKILL	Процесті құрту	Процестің аяқталуы
11	SIGSEGV	Сегментацияның қателігі (әдетте динамикалық жадының қате жұмысы кезінде пайда болады)	процестің аяқталуы және core файлының құрылуы
14	SIGALRM	Таймердің тайм-ауты басталуы; alarm() функциясымен генерацияланады	Процестің аяқталуы
15	SIGTERM	Процесстің аяқталуы	Бірдей
20	SIGCHLD	Аяқталу кезінде ата-аналарға топ-процеске берілуі	Еленбейді
30	SIGUSR1	Пайдаланушы сигнал	Процестің аяқталуы
31	SIGUSR2	Сондай	Бірдей



10.1-сурет. Сигналды
қабылдау кезінде өңдеушіге басқаруды беру



10.2-сурет. Сигналдардың типтерін бастапқы жағдайлары бойынша өңдеушіге басқаруды жіберу

Мұндай жағдайда, осындай сигналдар типтеріне пайдаланушы-өңдеуші анықталмаған делінеді. Сигналдың бұл типін процеспен қабылдау кезінде, операциялық жүйемен ұсынылатын, әдеттегідей өңдеуші-функциямен бағдарламалық код орындалуы басталады (10.2-сурет).

Сигналдар типінің көбісінде өңдеуші функция, сигналды қабылдаған процестің орындалуын әдеттегідей аяқтайды (10.1-кестені қараңыз). Сигналдардың кейбір типтері үшін пайдаланушы- өңдеуші мүлде анықталмауы мүмкін. Оған процесті аяқтаушы сигналдар жатады: SIGSTOP сигналы, уақытында енгізу/ шығару буферлерінің барлығы дұрыс тасталуы болатын процестің «жұмсақ» аяқталуын шақырушы; SIGKILL сигналы, процесс жұмысының тез арада апаттық аяқталуын шақырушы.

10.3.2. BASH СИГНАЛДАРЫ

BASH интерпретатордың көмегімен, белгілі PID сигналдарын тасымалдау үшін kill командасын қолдануға болады. Оны шақырудың параметрлері мынадай:

```
kill -<мнемоника немесе сигнал нөмері > <PID>
```

Мысалы, процеске PID = 1046 бірге SIGKILL сигналды тасымалдау үшін келесі сигнал нөмірін қолдануға болады:

```
kill -9 1046
```

Ал процеске PID = 1079 бірге, SIGINT сигналын тасымалдау үшін оның мнемоникасын пайдалануға болады:

```
kill -SIGINT 1079
```

Kill командасын шақырған кезде, нөмірді немесе жіберілетін сигналдың мнемоникасын көрсетпеуге де болады. Сонымен бірге, SIGTERM сигналы жіберіледі.

Процестің BASH тілінде соңғы іске асырылған тапсырмасынан PID алу үшін жүйелік айнымалы $! қолданылады. Белгісіз PID процесіне сигналды жіберу қажет болса, алайда оның атауы белгілі болса, туындайтын процестің іске асырылу нәтижесінде killall командасын қолдануға болады. killall командасы параметрлермен берілген сигналдарды барлық іске асырылған бағдарлама нәтижесінде туындаған процеске жібереді, killall командасының іске қосылуы параметрлермен беріледі. Команданы шақырту параметрі мынадай:

```
killall -<мнемоника немесе сигнал нөмері> <бағдарлама
аты>
```

Мысалы, timer бағдарламасының іске асырылуымен қосылған процеске SIGALRM сигналын жіберу үшін killall командасын келесідегідей қолдануға болады:

```
killall -ALRM timer
```

Killall командасын шақырту кезінде дәл осылай жіберілетін сигнал нөмірін көрсетпеуге де болады, сонымен қатар әдеттегідей SIGTERM сигналы жіберілетін болады.

10.3.3. СИГНАЛДАРМЕН ЖҰМЫС ЖАСАУ ҮШІН ЖҮЙЕЛІК ШАҚЫРУЛАР

UNIX-жүйелерде сигналдармен жұмыс жасау үшін жүйелік шақыртулардың интерфейсі анықталды. Жұмыс үшін ақпараттар

түрлері мен функция түпнұсқасы, signal.h тақырыптық файлдарда анықталған.

С-бағдарламасында сигналдарды беру үшін kill() жүйелік шақыртулары пайдаланылады. :

```
#include <signal.h>
int kill(pid_t pid, int signal_num);
```

Мұнда signal_num параметрі жіберілетін сигналдың атын береді. Егер pid параметр — оң сан болса сигнал PID процесіне беріледі, параметр мәніне тең. Егер pid параметрі нөлге тең болса EUID жіберуші сигналы EGID тең болса, онда сигнал барлық процеске жіберіледі. Егер EUID = 0 болса, яғни сигналдарды жіберу суперпайдаланушы root - тан жүреді, PID = 0 және PID = 1 процестерінен басқа, сигнал барлық процеске жіберіледі. Егер pid параметрі — теріс сан болса сигнал EGID параметрінің абсолютті мәніне тең болатын барлық процестерге жіберіледі.

Келесі бағдарлама kill() функциясының қолданылуына мысал келтіреді — бағдарлама SIGTERM сигналының көмегімен ата-ана-процесінің орындалуын еріксіз аяқтайтын, топ-процестердің пайда болуына себеп болады және осы туралы хабарлама шығарады:

```
#include <unistd.h>
#include <process.h> /* Linux - ке қажетсіз */
#include <stdio.h>
#include <sys/types.h>
#include <signal.h>

int main()
{
pid_t pid;
setvbuf(stdout,(char*)NULL,_IONBF,0);

switch (pid = fork() )
{
case -1:
perror("Unsuccesful fork");
_exit(1); break; case 0:
printf("Parent process with PID = %d\n", getppid());
kill( getppid(), SIGTERM);
sleep(1);
printf("Parent process terminated\n");
printf("New parent process have PID = %d\n",
```

```
getppid() ) ;
_exit(0);
break;
default:
while (1);   /* шексіз цикл */
             /* SIGTERM  қабылдауға дейін*/
}
}
```

Жоғарыда келтірілген бағдарламаға параметр ретінде берілген секунд санына (осы жағдайда —1 с) процесстің орындалуын тоқтата тұратын sleep() функциясы қолданылған. Бұл ата-ана-процесінің аяқталуға уақыттың жеткілікті қалуы үшін жасалған, одан кейін топ-процесінің PPID ауысуы болады. Sleep() функциясы және оны қолдану ерекшеліктері туралы бұдан әрі қарастыралатын болады.

Процестердің жұмысы нәтижесінде экранға мыналар шығады:

```
Parent process with PID = 1276 Terminated
Parent process terminated New parent process
have PID = 1
```

Экранға «Terminated» сөзін шығару SIGTERM сигналының әдепкі өңдеушілермен орындалған.

Бағдарламаға сигналды өңдеуші-функциясын беру үшін signal() жүйелік шақыруы қолданылады:

```
#include <signal.h>
void signal(int signal_num, void *handler_address);
```

Мұнда signal_num сигнал нөмірін береді, оны қабылдаған кезде процесс өңдеуші-функциясының орындалуын бастайды. Оның мекен-жайы handler_address параметрімен беріледі.

Жоғарыда айтылғандай, өңдеуші-функцияның тақырыбының форматы мынадай болады:

```
void handler(int sig_num);
```

мұнда sig_num параметрі арқылы өңдеуші-функцияға сигнал нөмірі беріледі, оны қабылдаған кезде, функция шақырылады.

Сигналдың өңдеушісін анықтайтын типтік бағдарламалар мынадай бейнеленеді:

```
#include <unistd.h>
```

```
#include <signal.h>
#include <stdio.h>

void handler( int sig_no )
{
printf("Signal with number %d caught\n", sig_no); _exit(
sig_no ) ;
}

int main()
{
signal( SIGALRM, &handler ) ; signal( SIGINT, &handler )
; while (1); return 0;

}
```

[Ctrl+C] батырмаларын басқан кезде бағдарлама келесіні

шығарады: `Signal with number 2 caught`
Себебі [Ctrl+C] үйлесімді батырмаларын басу ағымдағы процеске SIGINT сигналын жібереді.

Мұнда main()функциясы SIGALRM және SIGINT сигналдары үшін өңдеуші-функциясы анықталады, одан кейін бағдарлама сигналдың келуін күтетін шексіз циклге түседі. Екі сигналдың біреуін қабылдаған кезде handler() өңдеуші – функция шақырылады, ол экранға қабылданған сигналдың нөмірін шығарады және қабылданған сигналдың нөміріне тең қайтару кодымен процесті аяқтайды.

10.3.4. СИГНАЛ АЛМАСУДЫҢ УАҚЫТША СИПАТТАМАЛАРЫ

Соңғы мысалды талдау кезінде, егер SIGALRM немесе SIGINT сигналдары signal() функциясы осы сигналдарды өңдеушіні орнатқанға дейін процеспен қабылданған болса, онда handler() функциясы емес, ал әдеттегідей сигнал-өңдеуші шақырылатына назар аударғаныңыз жөн. Берілген нақты бағдарлама үшін бұл мәселе өте маңызды емес, себебі процестің іске асырылуы мен сигналды орнату аралығына кететін уақыт процестің іске асырылуы мен оған бірінші сигналдың жетуіне кететін уақыттан аз. Дегенмен егерде, сигналдарды орнатудың алдында бағдарлама есептеуші ресурстардың мәнін талап ететін кейбір әрекеттерді орындаса, демек, орындалуға кететін уақытты да, бірінші сигналды қабылдағанға дейін сигналдардың өңдеушілері қондырылатына да кепілдік бере алмайтын еді.

Процеске келіп түскен сигналдың процестің өзіне емес сыртқы ортаға байланысты болатындықтан, процестер арасындағы сигнал алмасудың уақытша сипаттамаларының есебі және уақыттың нақты сәтінде сигналдардың түсу мүмкіндігі — сигналдарды пайдаланушы бағдарламаларды жобалау кезінде есепке алынуы қажет негізгі сәттер болып табылады.

Процесс басқа жұмыспен айналысқан кезде, қолайсыз уақытта сигналдардың түсу мүмкіндігін төмендету, біріншіден, басқа процестермен (мүмкін болған кезде) сигналдарды жіберуі сәттерін нақты жоспарлау үшін екіншіден, сигналдарды қабылдау мүмкіншілігі көп болған уақытта, қабылдаушы-процестің орындалуын тоқтата тұруы үшін қажет. Егер процесс тоқтатылған күйде сигналды қабылдап алса, мұны ешқандай есептеуіш процесс үзе алмайды.

Процестің орындалуын екі әдіспен тоқтатуға болады. Бірінші процестің орындалуын тоқтатады және тоқтатылған жағдайдан шығу тек процес сигналды қабылдаған кезде мүмкін болады. Осындай кідіріс үшін pause() функция қолданылады, ал кідірту тәсілінің өзін, үзілістегі процесті орнату деп аталады. Екінші тәсіл уақыт аралығында берілетін процестің орындалуын тоқтатады. Процесс орындалуы осы аралықтың бітуімен немесе сигналды қабылдауымен жалғасады. Мұндай кідіріс үшін sleep() функциясы орнатылады, ал тәсілдің өзі «процесті ұйықтату» деген атауға ие болды.

Pause() және sleep() функцияларының мынадай прототиптері бар :

```
#include <signal.h>
void pause(void);
int sleep(int sleep_sec);
```

Pause() функциясы ешқандай параметрлерді қайтармайды және қабылдамайды. Функциямен орындалатын жалғыз әрекет — процеспен сигналдың қабылдауына дейін процестің орындалуын кідірту. Сигналды қабылдағаннан және өңдеуші аяқтағаннан кейін басқару pause()-н кейінгі командаға беріледі.

Sleep() функциясы ұзақтығы секундта sleep_sec параметрінің мәніне тең, уақыт аралығындағы процестің орындалуын кідіртеді. Процесс кідірісі кезінде сигнал қабылданған болса берілген уақыт аралығының аяқталуына дейін қалған толық секундқа тең мәнді қайтарады. Мысалы, егер уақыт аралығының аяқталуына 5,8 с қалса онда функция 5 қайтарады.

Егер уақыт интервалын нақтырақ беру талап етілсе usleep() функциясын қолдануға болады:

```
#include <signal.h>
```

```
void usleep(long sleep_usec);
```

usleep() функциясы sleep() функциясынан уақыт бірлігінің өлшенуімен ерекшеленеді —sleep_usec параметрі уақытты микросекундта береді. Sleep() функциясына қарағанда, usleep() функциясы - егер процесс usleep() функциясының жұмысы кезінде сигнал алған болса, онда интервал аяқталуына дейін қалған уақыт мәнін қайтармайды.

10.3.5. СИГНАЛДАРДЫ ӨҢДЕУШІМЕН БАСҚАРУ

Процеспен сигналды алған соң бұл сигналға сәйкес келетін- сигнал-өңдеуші кестесінің жолы өшіріледі. Егер процеспен сигналды қабылдаған соң, сигнал өңдеушісі қайта құрылмаса, онда сигналды аларда әдеттегідей өңдеуші шақырылады. Сигналды өңдеушіні қайта құру үшін әдетте сигналдың өңдеуші-функциясы денесіне signal() шақыру функциясын сыйғызады. Мысалы, келесі бағдарлама әр SIGINT сигналын алуы кезінде, сонымен қатар сигналды қабылдау үнемі күтетін «Сигнал қабылданды» жолын шығарады:

```
#include <signal.h>
#include <stdio.h>

void handler(int sig_no)
{
signal(sig_no, &handler);
printf("Signal caught\n");
}

int main()
{
signal( SIGINT, &handler);
while(1)
{
printf("Waiting for signal...\n");
pause();
}
return 0;
}
```

Бұл бағдарламаны іске қосарда (ағымдағы каталогтағы орындалатын файл — signal3) келесі міндеттер:

```
#!/bin/bash
./signal3 &
pid=$!
sleep 1
kill -SIGINT $pid
sleep 1
kill -SIGINT $pid
sleep 1
kill -SIGTERM $pid
```

экранға шығарылады:

```
Waiting for signal...
Signal caught
Waiting for signal...
Signal caught
Waiting for signal...
./signal3.sh: line 10: 1044 Terminated
```

Берілген бағдарлама SIGINT сигнал өңдеушісін — handler() функциясын орнатады, одан кейін шексіз циклге кіреді, ол өзінің орындалуын сигналды қабылдағанға дейін тоқтатады, бұл сәйкес келетін хабарламаны шығарады. Сигналды қабылдаумен қатар басқару handler() функциясына беріледі, ол пайдаланушыны сигналдың түсуін хабарлайтын хабарламаны шығарады және сигналдың өңделуін қалпына келтіреді. Осыдан кейін басқару pause() кейінгі командаға беріледі, яғни шексіз циклдің келесі итерациясы басталады.

Өңдеуші кестені, келесі сигналдың осы өңдеушімен өңделетіндей етіп қалпына келтіруі міндетті емес. Өңдеушілердің қалпына келуін процесстің сигналға реакциясы ағымдағы уақытпен алмасуы міндетті болатын жағдайларда пайдалануға болады.

Мысалы, келесі SIGINT сигналының тақ санын алуы кезінде бағдарлама A әрпін шақырады (бірінші, үшінші және т.б.) және SIGINT сигналының жұп санын алу кезінде B әрпін шақырады (екінші, төртінші және т.б.). Бұл әсерге, SIGINT өңдеуші сигналдың сатылы

ауысымымен ғана жетуге болады:

```
#include <signal.h>
#include <stdio.h>

void handler1(int sig_no);
void handler2(int sig_no);

void handler1(int sig_no)
{
signal(sig_no, &handler2);
printf("A\n");
}

void handler2(int sig_no)
{
signal(sig_no, &handler1);
printf("B\n");
}

int main()
{
signal(SIGINT, &handler1);
while (1)
pause();
return 0;
}
```

Келесі тапсырмада осы бағдарламаны орындау кезінде (бағдарламаның орындалатын файлдың атауы — signal4 деп тұжырымдалады):

```
#!/bin/bash
./a &
pid=$!
sleep 1
kill -SIGINT $pid
sleep 1
kill -SIGINT $pid
sleep 1
kill -SIGINT $pid
sleep 1
kill -SIGINT $pid
sleep 1
```

```
kill -SIGTERM $pid
```

экранға шығарылады:

```
A
B
A
B
./signal4.sh: line 14: 2452 Terminated
```

Жоғарыда қарастырылған сигналдар механизмінің уақытша шектеулері — пайдаланушы өңделуінің орнатылуына дейін сигналды қабылдау кезінде әдеттегідей өңдеушіні шақыру – тек бағдарламаның негізгі функцияларын орындау уақытында ғана емес, өңдеуші-функцияның орындалу уақытында да әрекет етеді.

Осылайша, егер өңделушінің бірінші командасымен сигнал өңделуін қалпына келтірсе (алдыңғы үлгілерде жасалғандай) өңдеушінің кез келген келесі команданы орындау сәтінде сигналды алу кезінде өңдеушінің өңделушінің орындалуы тоқтатылуына болады. Сигналды қабылдау сәтінде өңделуші қалпына келген болса, онда ол қайта жаңадан іске қосылады. Егер оның орындалу уақытында жаңа сигнал қабылданбаса, басқару сол өңдеуші-функциясына қайта келеді.

Мұндай тәртіп функцияның рекурсивтік шақыртуларымен ұқсас келеді, сонымен қатар мұндай «рекурсияның» тереңдігі, өңдеушілердің орындалу уақытында алынған сигналдардың санына тең болады. Әдебиетте әдетте, «өңдеушінің рекурсивтік шақырылуы» деген термин кездеседі. Мұндай атау тіпті дұрыс емес, себебі қарапайым рекурсиямен ешқандай ортақтығы жоқ.

Өңдеушінің соңғы жолына signal() шақырту функциясын сыйғыза отырып, өңдеушінің қалпына келтірсе, процестің күйі басқаша болады. Осылайша өңдеуші шақыртуларының қайталауынан құтылуға болады, бірақ процес оның өңделушімен қалпына келу сәтіне дейін, сигнал қабылдап алса, процестің орындалуын аяқтайтын әдеттегідей өңдеуші шақырылады.

Жоғарыда көрсетілген қиындықтардан құтылу үшін үш нұсқаның біреуін пайдалануға болады:

1) Көлемді және сигнал өңдеушінің бағдарламалық кодының қиындығын азайту, осылайша оның орындалу уақытын азайтамыз, демек, өңдеушінің орындалуы кезінде сигнал қабылдау мүмкіндігі;

2) Өңдеуші тәсілдері туралы қабылданған шешімдерді кейінге қалдыру және SIG IGN сигналдарын елемейтін тұрақты шамалардың көмегімен елемеу.

SIG_IGN тұрақты шамалары өңдеуші функцияның мекенжайы орнына, signal() функциясының параметрі ретінде ұсынылады. Сигналды елемеуді орнатқаннан кейін олар үшін ешқандай өңдеуші — әдеттегідей пайдаланушы да, өңдеуші де шақырылмайды. Осылайша, өзі қайталанатын шақырулардан және әдеттегідей өңдеушінің шақыруынан қорғай отырып, өңдеуші-функцияның басында сигналды елемеу режимін орнатуға болады, ал өңдеушіні функцияның аяғында қалпына келтіруге болады.

Типтік өңдеуші бұл жағдайда мынадай болады:

```
void handler(int sig_no)
{
signal( sig_no, SIG_IGN);
...
...
signal( sig_no, &handler);
}
```

3) Сигналды пердені пайдалану.

10.3.6. СИГНАЛДЫ ПЕРДЕЛЕР

Процестің сигналды перделері, процеспен қабылданған сигналдардың қандайы еленбейді және өңделмейтінін анықтайды. Құрылу кезінде процесс өзінің ата-анасының сигналды пердесін мұраға алады, бірақ та процестің өмірлік циклінің жүрісі кезінде, сигналды перде өзгеруі мүмкін.

Сигналды перденің күйін оқу немесе орнату үшін sigprocmask() функциясы қолданылады:

```
#include <signal.h>
int sigprocmask(int cmd, const sigset_t *new_mask,
sigset_t *old_mask);
```

cmd параметрі перденің астында орындалатын әрекеттерді береді, және келесі мәндерді қабылдайды:

- SIG_SETMASK — процестің сигналды пердесін *new_mask* параметрі ретінде жіберілетін перденің мәніне алмастырады*:*
- SIG_BLOCK — процесс пердесіне *new_mask* параметрі ретінде

берілетін пердеде көрсетілген сигналдарды қосады. Қосылған сигналдар процеспен еленбейді;

- SIG_UNBLOCK — *new_mask* параметрі ретінде берілген пердеде көрсетілген сигналдарды процесс пердесінен сигналдарды өшіреді. Өшірілген сигналдар процеспен еленбейді.

Сигналды перденің мәндері өзгерген болса, оның көне мәнін сақтап қалу қажет, әйтпесе, мәндері сақталған айнымалылар сілтеме бойынша old_mask параметрі ретінде жіберіледі. Егер көне мәнді сақтау қажеттілігі болмаса, бұл параметрдің мәні ретінде NULL жіберіледі. Айнымалы процесінде сигналды перденің мәнін сақтау талап етілсе, old_mask тәрізді жіберілімдер, оны өзгертпей, new_ mask параметрінің мәні ретінде NULL жіберіледі.

Процестің пердесін анықтайтын, айнымалы sigset_t типін қалыптастыру үшін үш функция қызмет етеді— sigemptyset(), sigaddset() и sigdelset().

```
#include <signal.h>
int sigemptyset( sigset_t *sigmask ) ;
int sigaddset( sigset_t *sigmask, const int
signal_num );
int sigdelset( sigset_t *sigmask, const int signal_num
);
```

Sigemptyset() функциясы sigmask параметрі секілді берілетін айнымалыда құлыпталған сигналдардың барлығын тазартады. Sigaddset() функциясы, signal_num нөмірлі сигналын перденің блокталған сигналдары тізіміне қосады, мұнда берілген sigmask параметрі ретінде айнымалы болады. Sigdelset() функциясы кері әрекетті орындайды — signal_num нөмірлі сигналын sigmask пердесінде блокталған сигналдарды тізімнен өшіреді. Сәтті орындалу жағдайында, бұл функциялар 0 қайтарылады, сәтсіз болған жағдайда (мысалы, sigmask нұсқауында қате болғаны кезінде немесе signal_num нөмірі қате болған жағдайда) -1 қайтарылады.

Сигналды перделердің қолданысын келесі мысалда қарастырамыз. Процесс SIGINT сигналын күтеді, оны қабылдағанда бұл типтің жаңа сигналдарын өңдейді, «Таймер қосылды» хабарламасын шығарады, 5 с күтеді, «Таймер тоқтатылды» хабарламасы шығады, SIGINT сигналының өңделуін қалпына келтіреді, оның өңделуін блоктан шығарады және тағыда сигнал келуін күтеді. Егер SIGINT сигналы 5 секундтық үзілісте келсе, ол еленбей қалады. Қабылданған сигналдың

үзілісі, осы жағдайда өңдеуші шақыруының қайталануынан немесе әдеттегідей өңдеушінің шақыруға мүмкіндік береді:

```
#include <signal.h>
#include <stdio.h>

sigset_t sigmask;
void handler(int sig_no)
{
sigemptyset(&sigmask ) ;

sigaddset(&sigmask, sig_no ) ;
sigprocmask(SIG_BLOCK, &sigmask, NULL);

printf("Timer started\n");

sleep(5);

printf("Timer stopped\n");
signal( sig_no, &handler ) ;
sigprocmask(SIG_UNBLOCK, &sigmask, NULL);
}
int main()
{
signal( SIGINT, &handler ) ; while (1)
{
pause();
}
}
```

Келесі міндеттерден бағдарламаларды (оны орындайтын файлдың атауы— signal5) іске асыру:

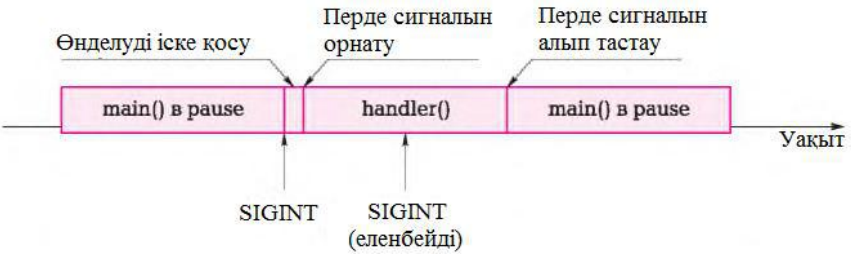
```
#!/bin/bash
./signal5   &
pid=$!
for i in 1 2 3 4 5 6 ; do
sleep 1
kill -SIGINT $pid
done
kill -SIGTERM $pid
```

экранға шығарылады:

```
Timer   started
Timer   stopped
Timer
```

```
started
./signal5.sh: line 11: 1276 Terminated
```

Sigemptyset() және sigaddset() функцияларының орындалу уақыты, өңдеудің басында өте аз. Дегенмен, егер де бұл функцияларды орындау кезінде сигналдар түсуі мүмкін болса, онда бұл функциялардың шақырылуын main() функциясының басына орнын ауыстыруға және sigaddset(&sigmask, SIGINT) шақыру көмегімен, SIGINT сигналын блоктаушы перденің қажетті мәнін қалыптастыруға болады. Сигнал нөмірі өңдеуші параметр арқылы емес, нақты берілетін болғандықтан, өзінің әмбебаптылығын құрбан етуі мүмкін.



10.3-сурет. Орнатылған сигналды перде кезінде процестің орындалуы

Егер процесс орындалуы жүрісін уақыт сызығында бейнелесек, 10.3- суретте көрсетілген бейнені алуға болады (екінші сигнал SIGINT еленбейді).

10.3.7. ТАЙМЕР

Сигналдарды пайдалауға негізделген жоғарыда қарастырылған барлық мысалдардың барлығында бір ортақ кемшілікке ие — олардың орындалу уақыты толығымен сыртқа ортаға тәуелді. Бұл бағдарламалардың орындалуын, SIGKILL немес SIGINT сигналдарын жіберіп басқа бағдарламалар немесе [Ctrl+C] басу арқылы пайдаланушы аяқтайды. Егер мұндай процесс пайдаланушының қатысуынсыз фондық режимде орындалатын болса, онда процестің аяқталуы тек басқа процестерге байланысты болады. Процестің өзі күту режимінде болып, өзінің орындалуын аяқтай алмайды.

Бұл мәселені шешу үшін UNIX-те жүйелік шақырту alarm() анықталады:

```
#include <signal.h>
unsigned int alarm(unsigned int time_interval);
```

Секундпен берілген, time_interval уақыт интервалының аяқталуы бойынша alarm() фукнциясын шақыртқаннан кейін ядро осы функцияны шақыртқан процеске SIGALRM сигналын жібереді.

Егер alarm() функциясын уақыт интервалы аяқталмай шақыртқан болса онда функция time_interval параметрімен берілген жаңа интервал қояды және бұрынғы интервалдың аяқталуына дейінгі қалған секунд санын қайтарады. Егер time_interval параметрінің мәні осы кезде нөлге тең болса, онда жіберілім болдырылмайды.

Alarm() функциясы берілген уақыт аралығының аяқталуына байланысты процессті дұрыс аяқтау үшін қолданылауы мүмкін. Ол үшін процестің орындалуын аяқтайтын SIGALRM сигналын анықтау және бағдарламаның орындалуы басында alarm() функциясын шақыру жеткілікті. Мысалы, іске қосылғаннан 20 сек өткеннен кейін аяқтау үшін функцияны қолданылатын бағдарламалар мыналар:

```
#include <stdio.h>
#include <signal.h>

void hdlrAlarm(int sig_no)
{
printf("Execution terminating\n"); exit(0);
}
void hdlrInt(int sig_no)
{
signal( sig_no, &hdlrInt ) ; printf("Signal %d
caught\n", sig_no);
}

int main()
{
signal( SIGINT, &hdlrInt); signal( SIGALRM, &hdlrAlarm);
printf("Timer started\n"); alarm(20); while (1)
{
printf("Waiting...\n");
pause();
}
return 0;
}
```

Тапсырмалардан бағдарламаны іске қосуы кезінде (signal6 орындалатын файлдың атауымен):

```
#!/bin/bash ./a & pid=$!
for i in 1 2 3 4 5 6 ;
do sleep 1
kill -SIGINT $pid done

wait $pid
```

экранға шығарылады:

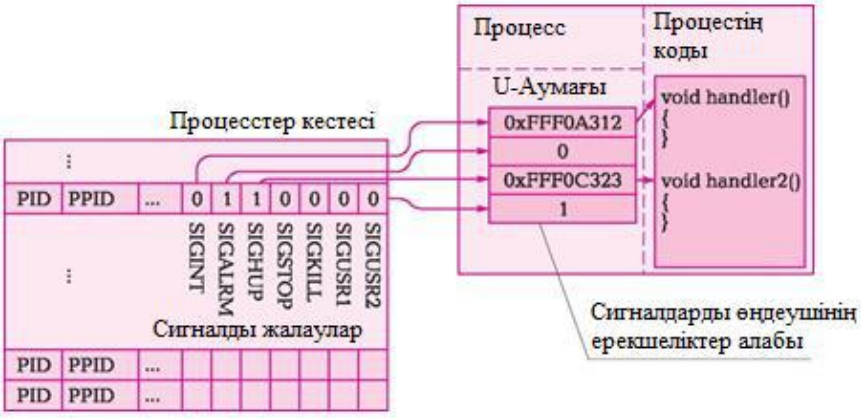
```
Timer started
Waiting...
Signal 2 caught
Waiting...
Signal 2 caught
Waiting...
Signal 2 caught
Waiting...
Signal 2 caught
Waiting...
Signal 2 caught
Waiting...
Signal 2 caught
Waiting...
Execution terminating
```

10.3.8. СИГНАЛДАРДЫ ЖОҒАЛТУ

Берілген тарауда қарастырылған сигналдар «сенімсіз сигналдар» деген атауға ие болады. Бұлай аталуының себебі, бұл сигналдардың қабылдаушы-процеске жеткізілуіне және оларды процеспен өңделуіне кепілдік жоқ. Олардың негізгі себебі, сигнал келіп түскеннен кейін өңдеуші-функциясының мекенжайына сәйкес келетін өңдеуші-кестелер жолдары беріледі, сондай-ақ бір уақытта бірнеше бірдей сигналдардың келуі кезінде процесс тек біреуін өңдейді. Олардың себебін анықтау үшін сигналдарды өңдеу кестесінің құрылымын және процеске сигналдың түсуін қарастырамыз.

Сигналды өңдеушілердің кестесі — UNIX-жүйелерінде жасалған абстрактілі құрылым екі жеке алаптардан тұрады: сигналдық жалаушалар алабы және сигналдардың ерекшеліктерін өңдеу алабы (10.4-сурет) [5].

Сигналдық жалаушылардың алабы процеске сәйкес келетін процестер кестесінің жолында сақталады, сигнал ерекшеліктерін өңдеуші алабы — процестің U-аумағында деп аталады, процестің жұмысы кезінде қолданылатын және қалыптасатын ақпараттарды құрайды. Әр сигнал бір типтің сигналына сәйкес келеді. Процеске сигнал жіберілген кезде, ядро сәйкес келетін жалаушаны оның процестер кестесінің жолына қояды.



10.4-сурет. Сигналдар өңдеушілер кестесінің құрылымы

Егер қабылдаушы – процесс күту жағдайында орналасса, ядро процесті белсенді жағдайға көшіреді және процестерді орындалу кезегіне қояды. Егер қабылдаушы-процесс белсенді болса, онда ядро орнатылған сигналдық жалаушасына сәйкес келетін сигналдар ерекшелігін өңдеу алабы элементінің мәнін оқи алады. Егер алап элементінің мәні нөлге тең болса процесс әдеттегідей өңделуде орындалады. Егер алап элементі бірді құраса сигнал еленбей қалады. Массив элементінің кез келген басқа мәні сигнал өңдеуші-функциясының мекен-жайы ретінде қолданылады. Соңғы жағдайда ядро басқаруды қабылдаушы-процестің ішіндегі өңдеуші-функциясына жібереді. Басқаруды өңдеуші-функцияға жіберместен бұрын ядро сигналдық жалаушаны алып тастайды және нөлдік мәнді сәйкес келетін сигналдарды өңдеуші алабының элементіне жазып алады. Осылайша, бірнеше бірдей сигналдарды сатылы түрде жіберсе қалғандары үшін әдеттегідей өңдеуші шақырылатын болады.

Бұдан бұрын айтылғандай, барлық тізбектеліп түсуші сигналдарды дұрыс өңдеу үшін ерекшеліктерді өңдеу алабының элементіне сәйкес өңдеуші – функциясының мекенжайын қалпына келтіру керек.

10.5- сурет. Әр түрлі уақыт сәтінде қабылдануы кезіндегі сигналға реакция

Дегенмен, процесс барлық сигналдарды өңдейді деген кепілдік беруге болмайды. Өңдеуші-функцияны шақыру сәтінен, өңдеу ерекшеліктері алабында мекенжайын қалпына келтіру моментіне дейінгі уақыт интервалында сигналды қабылдау кезінде (бұл интервал көп емес, бірақ нөлге тең ұзақтықта емес) процесс өңдеуді әдеттегідей орындайды.

Егер процесс ерекшеліктерді өңдеу алабында мекенжайды қалпына келтіру уақытында сигнал қабылдаса (қалпына келтіру процесі, сол сияқты көп уақытты алады), онда жарысатын жағдай туындайды — қай өңдеушінің шақырылатыны белгісіз: әдеттегідей өңдеуші немесе қалпына келтірілген пайдаланушының өңдеушісі (10.5-сурет).

Бұл ерекшеліктердің бәрі аз уақыт аралығында жіберілетін сигналдарды қабылдаушы қосымшаны әзірлегенде есепке алынады. Бұл тарауда қаралған сенімсіз сигналдардың кемшіліктерін болдырмас үшін UNIX-жүйесінде сенімді сигналдар механизмі пайда болады, қабылдаушы- процестермен сигналдарды өңдеу және жеткізу кепілдендірілген. Бұл механизмін қарастыру өзінің көлемінің шектеуінен берілген оқулықтың шегінен шығады онымен [5] және [16] оқулықпен танысуға болады.

10.3.9. ПРОЦЕССТЕРДІҢ ҮЙЛЕСІМДІЛІГІ

Себептері алдыңғы тарауларда көрсетілген сигналдарды жоғалту мәселелерін болдырмас үшін процестер аралығында сигналдар алмасуын, қабылдаушы-процестің оны дұрыс өңдейтін жағдайда болғанда, таратушы-процесс кезекті сигналды ертерек бергенге дейін ұйымдастыру қажет.

Осындай тәсілді іске асыруда мүмкін болатын әдістердің бірі - жіберуші-процесс сигналды уақыт аралығында береді, алдыңғы сигналдың өңделуін және өңделушінің қалпына келуін көп талап ететін қабылдаушы-процесс. Мысалы, келесі бағдарлама екі процесті іске асырады— оның іске асырылуы кезінде туындайтын аталық және fork() іске қосылумен туған еншілес. Еншілес процесс жіберуші рөліне түседі және аталық процеске SIGINT сигналын 1 секундта жіберілімдер арасында 10 интервалмен жібереді. Қабылдаушы (аталық процесс) сигналды қабылдауға жауап ретінде сигнал, олармен қабылданған SIGINT сигналдардың жалпы санын шығарады. Серияларды жіберу үшін 10 сигналдан SIGINT жіберушісі SIGTERM қабылдаушысына оны аяқтай отырып, сигнал жібереді, одан кейін өзінің орындалуын аяқтайды:

```
#include <process.h> /*  Linux –ке қажет емес*/
#include <sys/types.h>
#include <signal.h>
#include <stdio.h>

int num;

void handler(int sig_no)

{
signal(sig_no, &handler);
num++;
printf("%d\n", num);
}

int main()

{
pid_t pid;
int i; num
= 0;

setvbuf(stdout,(char*)NULL,_IONBF,0);

switch (pid=fork())

{
```

```
case -1:
perror("Fork error");
exit(1);
break; case 0:
sleep(5); /* аталыққа қою үшін уақыт береміз */
             /* өңдеуші */
for (i=0; i<10; i++)
{
kill( getppid(), SIGINT); /* Шлем SIGINT */ sleep(1); /*
қабылдаушының дайын болғанын күтеміз */
}
kill( getppid(), SIGTERM); /* қабылдауды аяқтаймыз */
_exit(0); /* жіберушіні аяқтаймыз */
break;
default:
signal( SIGINT, &handler);
while (1) pause();
_exit(0);
break;
}
return 0;
}
```

Бұл бағдарламаны орындаудың нәтижесінде экранға 1 ден 10 сандарының тізбегі шығады. Егер сигналдарды үздіксіз жіберілімдер арасында жіберсек, (sleep (1) шақыртуын алып), онда сигналдарды жоғалту қаупі артады, сондай ақ бұл қауіп артқан сайын жүйе процестің орындалуымен жүктеледі. Бұл жағдайда сигналдарды жоғалту 1 ден 10 ға дейін емес, ал аз санға дейін болуымен тұжырымдалады (мысалы, 8-ге дейін).

Бұл бағдарламаның негізгі кемшілігі, бұл процесс аралық өзара әрекеттесудің уақыты өте көп және уақыттың біраз бөлігін қабылдаушы-процесс сигнал күтумен алады.

Жұмыс уақытын азайту үшін және ақпарат санын арттыру үшін уақыт бірлігінде сигналдар көмегімен жіберілетін, олардың арасында екі бағытты сигналдар жіберілуін ұйымдастыра отырып, қабылдаушы мен жіберушінің орындалуын үйлестіреді. Сонымен қатар, жіберуші қабылдаушыға сигнал жібереді, ал қабылдаушы сигналдарды қабылдауға дайын және келесі сигналды өңдей алатынын хабарлаушы жауап-сигналдарын жібереді. Өндірілмейтін кідірістер мұндай схема кезінде, өзара әрекеттесу минимумға түседі.

Келесі үлгіде жіберуші-процес, қабылдаушы-процеске сигналдар

өңдеушісін орнату үшін (өңдеушілерді орнату үшін қажет, милисекунд бірлігінен азды- көпті нақты уақыт), сигналдар жіберу алдында 5 сек күтеді. Жіберуші мен қабылдаушы арасындағы ары қарай өзара әрекеттесуі екі бағытты — жіберуші SIGINT сигналын жібереді, қабылдаушы hdlrInt() функциясы арқылы сигналды өңдейді, өңдеуші қалпына келтіреді және жіберушіге SIGALRM қабылдаушының дайындығын куәландыратын сигнал –жауабын жібереді:

```
#include <process.h> /*  Linux-ке қажет емес */
#include <sys/types.h>
#include <signal.h>
#include <stdio.h>

int num;

void hdlrInt(int sig_no)
{
signal(sig_no, &hdlrInt);
num++;
printf("%d\n", num);
}

void hdlrAlarm(int sig_no)
{
signal(sig_no, &hdlrAlarm);
}

int main()
{
pid_t pid;
int i;

setvbuf(stdout,(char*)NULL,_IONBF,0);

switch ( pid = fork() )
{
case -1:
perror("Fork error\n");
_exit(1); break; case 0:
signal(SIGALRM, &hdlrAlarm);
sleep(5); /* қабылдаушының инилизациясын күтеді */
for (i=0; i<10; i++)
{
kill(getppid(), SIGINT); /* SIGINT жібереміз */
pause(); /* Ждем SIGALRM */
```

```
}
kill(getppid(), SIGTERM); /* қабылдауды аяқтаймыз */
_exit(0); /* жіберуді аяқтаймыз */
break;
default:
signal(SIGINT, &hdlrInt);
while (1)
{
pause(); /* Ждем SIGINT */
kill(pid, SIGALRM); /* hdlrInt-тен қайту */
                        /* және жауап жібереміз */
}
_exit(0);
break;
}
return 0;
}
```

Келтірілген үлгі әдейі қарапайымдандырылған. Келесіге назар аударған жөн, қабылдаушы да жіберуші де тек SIGALRM және SIGINT сигналдарымен ғана емес, кез келген сигналмен үзіліс жағдайынан (pause() функциясы кезінде) шығып кетуі мүмкін. Дұрыс жұмыс жасауы үшін қалған сигналдар сигналды перденің көмегімен не SIG_IGN тұрақты шамалардың көмегімен еленбеуі қажет.

10.4. ХАБАРЛАМАЛАР

Бұдан бұрын қарастырылған сигналдар механизмі процестер арасындағы ақпарат алмасуды қажет етпейтін жағдайларда процесс аралық өзара әрекеттесуді ұйымдастыруда ыңғайлы. Мысалы, өзара әрекеттесуші процестер мәтіндік ақпарат алмасқан жағдайда, сигналдар нашар қолданылады.

Дегенмен, сигналдар мына жағдайда қолданылуы мүмкін, мысалы әр символ нақты сигналдар тізбегімен кодталады, жіберілімнің басы және аяғы нақты типтің сигналдарымен белгіленеді. Мұндай тәсілдің ыңғайлылығы бір символды жіберу үшін сигналдардың үлкен саны жеткілікті (латын әрпін жібері үшін — символ жіберілімінң басы және аяғын бейнелейтін төртеу шамасында), ал бұл сигналдар жоғалған кезде, ақпараттардың жоғалуымен байланысты. Одан басқа, мұндай алмасу тәсілі процестің екі ақпарат алмасушысы да бір уақытта жұмыс жасағанда ғана қолданылады.

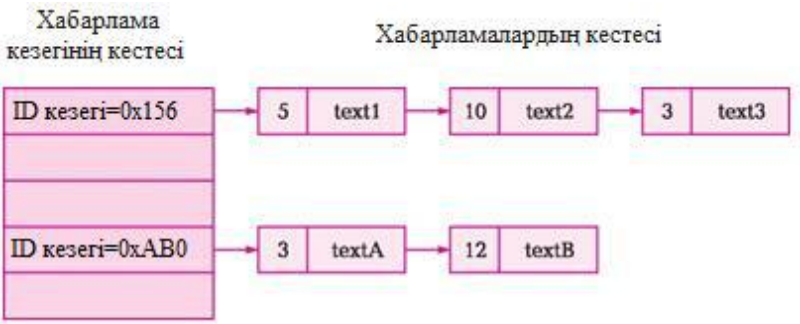
Процестер арасында ақпарат алмасу үшін (мүмкін, әр түрлі

уақыттарда орындалатындар) UNIX-жүйелерінде хабарлама механизмі бар.

Бұл механизмнің жұмысы, ортақ қолжетімді пошталық жәшікті еске түсіреді. Адамдар бұл жәшікке конвертке мекен-жай атын жазып, хаттарын сала алады, ал мекенжайлар ара-тұра пошталық жәшіктің мазмұнын тексереді, оларды мекенжай аттарымен қондыра отырып, өзіне арналған хатты алады. Сонымен қатар жәшікке хатты салуда, тізбектеулі сақталады, егер адамға бірнеше хат келген болса, басында ол ең бірінші келгендерін оқиды, одан кейін жаңаларына өтеді. Хаттардың мекенжай бойынша бөлінгенінен басқа, хаттардың әр түрлі жәшік бойынша бөлінуі мүмкін, мысалы бір жәшік тек екі адамның сұхбаттасуына арналуы мүмкін.

Хабарлама механизмі осы тәрізді жұмыс жасайды, операциялық жүйе - хабарламалар кезегі деп аталатын - хабарламалар тізбегі сақталатын арнайы жадтың саласын көрсетеді, мұнда процестер пошталық жәшік сияқты, өзінің хабарламаларын сыйдыра алады. Әр хабарлама, жіберілетін ақпаратты таситын, сандық идентификатор мен ақпараттар блогынан тұрады. Ақпараттарды кезекке орналастыру және оларды кезектен алып тастау FIFO (first in — first out) принципі бойынша жүреді, яғни кезектен ең көне хабарлама алынып тастайды. Дегенмен тізімдегі әр хабарлама идентификаторға ие, ол кезектегі барлық хабарламалардың ішіндегі ең көне болуы міндетті емес, идентификаторы берілген көне хабарламалар алынады (10.6 - сурет).

Осылайша, операциялық жүйе хабарламалардың сақталуына кепілдік береді және кезек түсу ретімен олардың өңделуіне мүмкіндік береді. Дегенмен хабарламалардың өзіне арналған емес, басқа процеспен кезектен алынуына кепілдік бермейді. Бұл алынатын хабарламаның идентификаторы кабылдаушы – процесті беруімен байланысты.



10.6-сурет. Хабарламалар кезегінің құрылымы

Хабарламалар кезегі және ондағы хабарламалар, оны пайдаланатын процестерге тәуелді емес. Осылайша, жөнелтуші процесі хабарламаны кезекке қояды және өзінің орындалуын тоқтататын жағдайлар болуы мүмкін. Қабылдаушы-процесс, жөнелтуші-процестің қолжетімсіз болуына қарамастан, бұл хабарламаны қабылдап алады.

Әр хабарлама кезегі, кезекке хабарлама қоятын және бір-біріне тәуелсіз кезектен алып тастайтын, бірнеше процеспен қолданылуы мүмкін. Сонымен қатар процестің әрі өзінің идентификатор жинағын қолданады. Дегенмен , егер бірнеше процесс бір идентификаторды қолданатын болса, әртүрлі жіберушілерден келген хабарламалар араласып кетуі мүмкін.

Бірнеше процестің, бір уақытта кезекке бірдей идентификаторлы екі хабарламаны сыйғызатын жағдай да болуы мүмкін. Мұнда хабарламаларды кезекке сыйғызу реті ерікті болады және операциялық жүйенің ядросының жасалуына байланысты болады.

Бірнеше процестің бір уақытта кезектен, берілген идентификаторлы соңғы хабарламаны алуына тырысатын жағдайларда болуы мүмкін. Хабарламаны процеске беру реті анықталмаған және ОЖ ядросының жасалуына байланысты.

Әдетте мұндай жағдайлар өзімен күрделі мәселелерді ұсынбайды, себебі бір кезекте ақпараттар алмасатын процестер жұбын байқап жобаласа, байланыспайтын көптеген идентификаторларды қолданады. Құрылған кезек, ОЖ процесінің құрылуымен және осыған құқығы бар

процеспен өшірілу сәтіне дейін болады немесе ОЖ жұмысының аяқталуына дейін. Кезек өмірінің уақыты, оны құрған процесс өмірі уақытынан ұзақ болуы мүмкін.

Хабарламалар кезегінің параметрі, kemel.msgmni, kemel.msgmax және kemel.msgmnb параметрлерімен анықталады. Параметр kernel. msgmni хабарламалар кезегінде идентификатордың максималды санын, kemel.msgmax — байтта хабарлама максималды өлшемін анықтайды, ал kemel.msgmnb байтта кезектің максималды өлшемін береді. Бұл параметрлерді /proc файлдық жүйеде орналасқан файлдарды пайдаланып, өзгертуге болады. Ол үшін мына командаларды қолдануға болады:

```
echo 2048 > /proc/sys/kernel/msgmax
echo 4096 > /proc/sys/kernel/msgmni
```

Бұл өзгерістерді, оның қайта жүктелімінен кейін де, жүйеде сақтау үшін бұл параметрлердің мәндерін қолмен /etc/sysctl.conf файлына енгізу керек. Бұл файлды «kemel.msgmnb=8192» түрінде жолға түсіру керек немесе sysctl -w <paramer> = <value> командасын қолдану керек, мысалы sysctl -w kernel.msgmnb=8192.

Хабарламалар кезегі параметріндегі ағымдағы мәндері туралы ақпараттарды, ipcs –lq командасының көмегімен алуға болады:

```
$ ipcs -lq

    Messages: Limits ------------
max queues system wide = 496                // MSGMNI
max size of message (bytes) = 8192          // MSGMAX
default max size of queue (bytes) = 16384   // MSGMNB
```

Қазіргі сәтте пайда болған кезектерді қарау үшін жүйеде ipcs командасы қолданылады. Параметрсіз іске асатын, командалар жалпы жадының барлық пайда болған бөліктерін, семафорды және кезектерді шығарады. Пайда болған кезектерді қарау үшін командалық жолдан -q параметрімен ipcs іске асыру қажет:

```
$ ipcs -q

    Message Queues ------------
key          msqid  owner   perms   used-bytes   messages
0x000001f4   0      admin   644     20           1
0x1cda78da   486    root    600     18920        211
```

10.4.1. Unix хабарламаларға арналған ақпараттар құрылымы

Жүйедегі хабарламалар кезегін сипаттайтын барлық қызмет ақпараттары ОЖ ядро жадында сақталған хабарламалар кезегінің кестесінде болады. Мұндай кестеде әр жазба бір кезекті бейнелейді және келесі ақпаратты құрайды:

- кезек идентификаторы — кезекті бір мәнді сәйкестендіретін бүтін сан. Идентификатор оны процеспен жасайтын кезекке қосылады; ақпарат алмасу үшін кезекпен жұмыс істейтін процестер осы идентификаторды кезекке қолжеткізу үшін қолдануы мүмкін.
- UID және GID кезек құрушылар —EUID кезек құрушы UID сәйкес келетін процестер кезекті басқара алады: оның параметрін өзгерте алады немесе өшіре алады;
- PID процессі хабарламаны және осы оқиғаның уақытын соңғы кезекке қойған;
- PID процесс кезектегі хабарламаны және осы оқиға уақытын соңғы оқыған;
- Ядро жад аумағын көрсетуші, мұнда кезекте сақталатын хабарламалармен сызықтық тізім тұрады. Мұндай сызықтық тізімнің әр элементі ақпараттық хабарламалар мен оның идентификаторынан тұрады. Одан басқа, тізім элементі қызмет ақпаратын құрайды — байттағы хабарлама өлшемі және сызықтық тізімнің елесі элементіне көрсетуші. Тізімдегі элементтер тізбегі кезектегі хабарлама тізбегін анықтайды.

Файлдарға сияқты кезекке қолжетімділікті шектеу үшін пайдаланушы-иесі және пайдаланушы-топтар құқығын беру тәсілі қолданылады. Кезектердің максималды саны, кезектің максималды өлшемі және хабарламалар ОЖ ядросымен анықталатын тұрақты шамалармен беріледі.

Хабарламаларды кезекке қойған кезде сызықтық тізімнің жаңа элементі пайда болады, оған хабарламаның деректері және идентификатор сыяды. Хабарлама деректері мен идентификатор процестің мекенжай кеңістігінен ядро жадының мекенжай кеңістігіне көшірілетін болғандықтан, жөнелтуші-процес өзінің орындалуын кез келген сәтте тоқтатуы мүмкін — кезектегі хабарламалар тиіспеген күйде қалады.

10.4.2. Хабарламалармен жұмыс жасауға арналған жүйелік шақыртулар

UNIX хабарламаларымен жұмыс жасау үшін интерфейс файлдармен жұмыс істейтін интерфейсті еске түсіреді — мұнда объектілерді жасау үшін функциялар, одан ақпараттарды жазу мен оқу, оның жағдайын басқару бар. Обьекті ретіндегі файлдар үшін екілік жолдар немесе мәтіндік файлдардың ақпараттары рөліндегі хабарламалар түседі. Файлдың күйі — оған қолжеткізу құқығы мен режимі. Обьектілер рөліндегі хабарламалар үшін хабарламалардың кезегі, ақпараттар рөліне — хабарламалардың өзі түседі, кезектің күйі оған қол жеткізудің құқығы мен режимін анықтайды.

Осыған сәйкес, кезекпен жұмыс жасау үшін төрт негізгі функциялар бар. Бұл функциялардың барлығы тақырыптық файлдардың sys/types.h, sys/ipc.h және sys/msg.h қосылуын талап етеді:

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
int msgget( key_t key, int flag);
```

Кезекті құру және ашу үшін msgget() функциясы қызмет етеді. Бұл функцияның key параметрі ретінде, бірегей сандық идентификатор кезегі беріледі (файлдармен жұмыс жасауға арналған fopen() тәрізді функцияның аналогы). Егер key параметрі ретінде IPC_PRIVATE тұрақты шамасы берілсе, онда оны құрған процеспен ғана қолданылатын кезек жасалады. Flag параметрі кезекті құруға арналған параметрлерді береді. Параметр ретінде IPC_CREAT тұрақты шамасын берілсе, онда msgget шақыртуы бірегей идентификаторлы берілген кезекті құрады. Кезекке қолжетімділік құқығын анықтау үшін IPC_CREAT тұрақты шамасын қолжетімділік құқығы жазбасының сандық формасын біріктіреді, мынадай:

```
m_id= msgget (500, IPC_CREAT | 0644);
```

Сонымен қатар r құқығы кезектен хабрламаларды оқу мүмкіндігін береді, w құқығы — кезекке хабарламаларды сыйғызады, ал x құқығы — кезектегі параметрлерді басқаруға мүмкіндік береді.

Msgget() функциясы кезектің дескрипторын қайтарады, одан кейін ол жіберілімдер және хабарламаларды қабылдау функциясымен қолданылады (файлдармен жұмыс жасау үшін дескриптордың аналогы, FILE* типтік файлдық дескрипторы болып табылады).

Егер msgget() функциясының параметрі ретінде 0 жіберуге болады, ол идентификатормен берілген кезекті ашады және оның дескрипторын қайтарады. Егер кезекті құруды IPC_EXCL тұрақты шамасымен біріктірсе, және кезекті құру және ашу сәтсіз болса, функция -1 қайтарады.

Хабарламалар құрылым ретінде ұсынылып, мына параметрлерге ие болады:

```
#define LENGTH 25 6

struct message
{
long m_type;
char m_text[LENGTH];

}
```

Ұзақ бүтін m_type хабарлама түріндегі идентификаторды, m_text символдар алабын — хабарламалар мәтінін береді. Бағдарламалармен өңделген хабарламалардың, максималды мүмкін болатын ұзындығын беретін LENGTH тұрақты шамасы (қаралатын мысалда — 256 символ), кезекке сыйған хабарламалардың мүмкін болатын ұзындығын беретін MSGMAX жүйелік тұрақты шамасына тең немесе аз болуы керек:

```
int msgsnd (int msgfd, void *msgPtr, int len, int
flag);
```

Кезекке хабарламаларды жіберу үшін msgsnd() функциясы қызмет етеді, ол msgget() шақыртуының нәтижесінде болатын, msgfd параметрі ретінде кезектің дескрипторын қабылдайды, msgPtr шақыртуы ретінде — хабарламалардан тұратын ақпараттар құрылымын көрсетеді; len параметрі хабарламаның m_text элементінің ұзындығын береді. Егер flag параметрі ретінде 0 берсе, онда процестің орындалуын, хабарламаны кезекке сәтті орналастырған сәтке дейін тоқтата тұруға болады дегенді білдіреді. Мұндай кідірістер, хабарлама кезегі, msgsnd() шақыртуы кезінде толып және кезектен оның орнын босатуы үшін хабарламалардың, жеткілікті алуын күту қажеттілігімен байланысты болуы мүмкін. Егер flag параметрі ретінде IPC_NOWAIT тұрақты шамасын көрсетсе, онда операциялардың сәтсіз орындалуындағыдай, функция орындалуы тоқтатылады және -1 қайтарады.

Келесі бағдарлама кезек құрады және оған хабарлама жібереді:

```
#include <stdio.h>
#include <string.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include <sys/types.h>

struct message
```

```
{
long length;
char text[256];
} my_msg = { 25, "Sample text message" };


int main()
{
int fd;
fd = msgget(500, IPC_CREAT | IPC_EXCL |0644 );

if (fd != -1 )
{
msgsnd(fd, &my_msg, strlen(my_msg.text)+1,
        IPC_NOWAIT);
}
return 0;
}
```

Келтірілген үлгіде, msgget() функциясының сәтісз қайтару кодының қарапайым өңделуі енгізілген — кезекті құру мүмкін болмаған жағдайда (мысалы, кезекті сақтауға арналған жүйелік жады облысы толып қалса немесе мұндай сандық идентификаторлы кезек болған болса) хабарлама кезекке жіберілмейді.

Кезектегі хабарламаны алу үшін msgrcv() функциясы қызмет етеді:

```
int msgrcv(int msgfd, void *msgPtr, int len, int mtype,
int flag);
```

Msgfd параметрі ретінде, оған кезектің дескрипторы беріледі; параметр msgPtr, түскен хабарламаны сыйғызатын, жады облысының көрсеткішін анықтайды; параметр len, қабылданатын хабарламаның максималды мүмкін болатын ұзындығын береді. Mtype мәні қабылданатын хабарламаның типін береді. Бұл параметр келесі мәндерді қабылдайды:

- 0 — кезектен кез келген типтен ең көне хабарлама алынатын болады;
- Оң бүтін сан — кезектен, осы санға тең типті, ең көне хабарлама алынатын болады;
- Теріс бүтін сан — кезектен, осы санға тең немесе кіші типті ең көне хабарлама алынатын болады. Егер кезекте, осы белгіні қанағаттандыратын бірден көп хабарлама болса, типтің аз мәніне ие хабарлама қабылданатын болады.

Flag параметрі ретінде 0 көрсетуге болады, бұл жағдайда кезектегі

хабарламалардың жоқтығы кезінде, процесс хабарлама алғанға дейін өзінің орындалуын тоқтатады. Егер параметр ретінде, IPC_NOWAIT тұрақты шамасын көрсетсе, онда кезектегі қабылдауға арналған қол жетімді хабарламалар болмаған жағдайда, процесс өзінің орындалуын жалғастырады. Функция хабарламаны сәтті қабылдаған кезде немесе -1 сәтсіз болғанда, мысалы хабарлама ұзындығы, len параметрімен берген ұзындықтан асып кетсе хабарламаның мәтіндік бөлігінің байттар санын қайтарады. Мұндай жағдайдың алдын алу үшін және алғашқы len символдарын қабылдау үшін flag параметрі ретінде MSG_NOERROR тұрақты шамасын көрсетуге болады. Барлық көрсетілген тұрақты шамаларды ИЛИ операциясының топтары арқылы біріктіруге болады.

Келесі бағдарламаға алдыңғы бағдарламаның кезекке салған хабарламалардын қабылдайды және оның мәтіндік бөлігін экранға шығарады:

```
#include <stdio.h>
#include <string.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include <sys/types.h>

struct message
{
long length;
char text[256];
} my_msg = { 0, "---------------" };

int main()
{
int fd, rcv_res;
fd = msgget(500, IPC_EXCL ) ;
if (fd != -1)
{
rcv_msg = msgrcv(fd, &my_msg, 1024, MSG_NOERROR); if
(rcv_msg != 1)
{
printf("%s\n", &my_msg.text)
}
}
return 0;
}
```

Хабарламаны сәтсіз қабылдаған кезде бағдарлама экранға ештеңе

шығармайды.

Кезекті басқару үшін және оның параметрлерін қабылдау үшін msgctl() функциясы қызмет етеді:

```
int msgctl(int msgfd, int cmd, struct msqid_ds
*mbufPtr);
```

Msgfd параметрі ретінде, оған кезектің дескрипторы беріледі; mbufPtr параметрі кезекті басқару параметрін береді; параметр cmd, кезекте орындалатын команданы береді; Параметр келесі мәндерді қабылдай алады:

- IPC_STAT — кезектің басқаратын параметрлерін құрылымға көшіру, көрсеткіш mbufPtr параметрімен беріледі;
- IPC_SET — құрылымда болатын параметрлерді кезекпен алмастыру, көрсеткіші mbufPtr параметрімен беріледі бұл операцияны сәтті орындау үшін пайдаланушы – root пайдаланушы болуы керек, немесе құрушы, немесе кезектің иесі тағайындауымен болуы мүмкін;
- IPC_RMID — кезекті жүйеден өшіру. Бұл операцияны сәтті орындау үшін пайдаланушы – root пайдаланушы болуы керек, немесе құрушы, немесе кезектің иесі тағайындауымен болуы мүмкін. Бұл жағдайда, mbufPtr параметрі ретінде NULL беріледі.

- Msqid_ds құрылымы хабарлама кезегінің келесі параметрлерін береді:

```
msqid ds {
ipc perm msg perm; /* кезекке қол жетімділік
msg *msg first;     /*    кезектегі    алғашқы
                хабарламаны көрсетуші */
msg *msg last ;     /*     кезектегі     соңғы*/
                хабарламаны көрсетуші */
msg stime;      /* msgsnd соңғы шақыртуының
                уақыты*/
msg rtime;      /* msgrcv соңғы шақыртылған
                уақыты*/
msg ctime;      /* кезектегі соңғы өзгерістің
                уақыты */
msg cbytes;     /* кезектегі барлық
                хабарламалардың сомалық өлшемі
msg qnum;       /* кезектегі хабарламалар саны
                */
msg qbytes;     /* кезектің байтпен максималды
                өлшемі */
msg lspid;      /* кезектегі мәндерді оқитын
                соңғы процесс PID */

msg lrpid;      /* кезекке мәндерді жазатын PID
                соңғы процесі *
                                              /
```

struct
struct
struct
struct
time_t

```
time_t

time_t

ushort

ushort

ushort

ushort


ushort
```

Келесі бағдарлама екі процесс аралығындағы хабарламалардың көмегімен ақпарат алмасуды безендіреді. Бағдарлама екі процесті туындатады — аталық процесс және топ-процесс. Аталық топқа латын әріптері түріндегі хабарлама жібереді, әр хабарламаның типі — әріп нөмірі. Топтары бұл хабарламаларды кері ретпен қабылдайды. Қолданылатын кезектің идентификаторы PID аталық-процесіне сәйкес келеді:

```
#include <unistd.h>
#include <stdio.h>
#include <sys/types.h>
#include <signal.h>
#include <sys/ipc.h>
#include <sys/types.h>
#include <sys/msg.h>

struct message
{
long type;
char text[10];
};

int main()
{
pid_t pid;
int qId;
char i;
int *status;
struct message my_message;
setvbuf(stdout,(char*)NULL,_IONBF,0);

switch (pid = fork() )
{
case -1:
perror("Bad fork\n");
_exit(1);

break;

case 0:

/* топ денесі */
/* аталықтың кезекті толтырғанын күтеміз */
sleep(10);
/* құрылған кезектерді ашу */
qId = msgget(getppid(), 0 ) ;
```

```
if (qId == -1)
{
printf("Unable to open queue\n");
_exit(0);
}
printf("r> "); for (i=26;i>0;i--)
{
/* кері ретпен хабарламаны қабылдау */
msgrcv(qId,&my_message, 2, i, MSG_NOERROR);
printf ("%d:%s ", my_message.type,
        &my_message.text);
}
printf("\n\n\n");
/* кезекті өшіру */
msgctl(qId, IPC_RMID, NULL);
_exit(0);
break;
default:
/* аталық денесі */
printf("Queue ID: %d\n",getpid());
/* кезекті құру */
qId = msgget(getpid(), IPC_CREAT | 0666 );
if (qId == -1)
{
printf("Unable to create queue\n");
kill(pid, SIGKILL);
_exit(0);
}
printf("s> ");
for (i='A'; i<='Z'; i++)
{
/* хабарламаны құру*/
my_message.type = i - 'A' + 1;
my_message.text[0] = i;
my_message.text[1] = 0;
print ("%d:%s ", my_message.type,
       &my_message.text);
/* хабарламаны кезекке салу */
msgsnd(qId, &my_message, 2, 0);
}
printf("\n\n\n");
```

```
/* топтың аяқталуын күту */
wait(&status);
return 0;
}
}
```

Бұл жұмыстардың нәтижесінде экранға келесі мәтін шығады:

```
Queue ID: 3952
s> 1:A 2:B 3:C 4:D 5:E 6:F 7:G 8:H 9:I 10:J
11:K 12:L 13:M 14:N 15:O 16:P 17:Q 18:R
19:S 20:T 21:U 22:V 23:W 24:X 25:Y 26:Z

r> 2 6:Z 25:Y 24:X 23:W 22:V 21:U 20:T 19:S
18:R 17:Q 16:P 15:O 14:N 13:M 12:L 11:K
10:J 9:I 8:H 7:G 6:F 5:E 4:D 3:C 2:B 1:A
```

«s>» жолынан аталықтармен жіберілген, «r>» жолынан — ұрақтармен қабылданатын хабарламалар басталады. Әрі қарай экранға идентификатор - мәтін жұбы, хабарламаны кезекке салынуы немесе алынуы шығады.

10.5. СЕМАФОРЛАР

10.5.1. Негізгі түсінік

Процесс аралық өзара әрекеттесуі кезінде ең маңызды міндеттердің бірі — процесстердің орындалуын біріктіру. Біріктіруде бірінші болып, бірнеше процестердің паралельді орындалуы деп түсіндіріледі, қалған процестердің дайын болатын жағдайы келу сәтіне дейін кодтың нақты бөліктерінің орындалуы жүрмейді.

Мысалы, матрицаларды минор бойынша анықтаушының параллельді есептеуі кезінде әр минор өзінің процесімен есептеледі, минордың үлкен реттелуге өтуі барлық процестер өзінің есептеулерін аяқтап және нәтиже алғаннан кейін мүмкін болады. Басқа процестердің дайындық күйі дайын болуы туралы хабарламаны жіберумен анықталады. Біріктіруді талап ететін басқа мысал, бірнеше параллельді жұмыс істейтін процестердің бәрі бір бөлінбейтін ресурсқа қол жеткізуі

болып табылады. Бұл жағдайда, біріктірілу дегеніміз бір уақыт сәтінде ресурсқа қол жеткізуді тек бір процестің алуы.

Біріктіру механизмдерінің ең көп қолданылатыны — семафорлар көмегімен біріктірілу.

Қарапайым семафор өзімен қолжетімділік тыйымына сәйкес келетін 0 мәнді жалаушаны ұсынады, ал 1 — қолжетімділікке рұқсат. Ресурсқа қол жеткізуден бұрын, процесс семафордың мәнін тексеру керек және егер қолжеткізуге рұқсат болса, ресурсқа басқа процестердің қол жеткізуін блоктау үшін семафорға 0 мәнін орнату қажет. Ресурсты пайдалану аяқталған соң процес, қайтадан семафорға 0 мәнін орнатады. Мұндай семафор *«бинарлық семафор»* деген атауға ие болды.

Бинарлы семафордан басқа, семафор-есептегіштер бар. Мұндай семафорлардың мәні теріс емес бүтін санды ұсынады. Мұндай семафорлардан екі операция анықталған. V операциясы (голл.Verhogen, жиі wait дегенді білдіреді) егер оның мәні аз немесе нөлге тең болса, процесс орындалуын тоқтатады немесе қарсы жағдайды семафордың мәнін бірлікке азайтады. Операция P (голл. Prolagen, post дегенді білдіреді) семафор мәнін бірлікке арттырады және мәндерді қайта тексеруді шақыра отырып, мәндердің артуынан процестердің тоқталуын хабарлайды.

Семафор-есептегіштер бір типті ресурстардың шектеулі санына бірігіп, қол жеткізу үшін қызмет етеді мысалы бір компьютерге қосылған теруші құрылғылар үшін. Қолжетімді құрылғылар саны, семафор-есептегіштің бастапқы мәні ретінде жазылады.

Құрылғыларды біреуін қолданбастан бұрын, процесс семафорларға V операциясын орындайды. Егер тек бір қолжетімді құрылғы қалған жағдайда, есептегіштің мәні бірлікке азаяды, осылайша қолжетімді құрылғының азайғанын бейнелейді. Егер бірде бір қолжетімді құрылғы қалмаған болса, процесс құрылғы босағанша өзінің орындалуын тоқтатады және есептегіш мәні оң болмайды. Себебі құрылғылардың босауын күтетін процестер бірнеше болуы мүмкін, онда олар кезекті және құрылғыға қол жеткізуді ұйымдастырады, есептегіш мәнінің артуы кезінде, кезектегі алғашқы процесті алады.

Ресурстардың босатылуынан кейін (біздің жағдайда — теруші құрылғы) процесс семафорға P операциясын орындайды, осылайша есептегіш мәнін бірлікке арттырады.

Семафор-есептегіштің қалыпты жағдайы үшін UNIX-жүйелерінде келесі шарттар орындалады:

- Семафордың мәні әртүрлі процестерге қолжетімді болуы керек, сондықтан процестің емес ОЖ ядросы мекенжай кеңістігінде болуы керек;
- Тексеру операциясы және семафор мәнінің өзгеруі, басқа процестермен үзілмейтін бір атомаралық операция түрінде іске асуы қажет. Әйтпесе, тексеріс пен семафор мәнін азайту арасында процестердің тоқталуы болуы мүмкін, бұл семафорды күтпеген жағдайға итермелеуі мүмкін.

10.5.2. Семаформен жұмыс жасауға арналған жүйелік шақыртулар

UNIX-жүйелерінде семаформен жұмыс жасауға арналған бірнеше интерфейстер бар. Бұл тарауда қарастырылатын басқаларынан бұрын пайда болған және «System V семафорлы интерфейс» атауына ие болған. Мұнда бір бүтін болып қарастырылатын, семафор- есептегіш жинағы ұғымы қолданылады. Жинақтың әр элементі жеке семафор-есептегіш ұсынады, сонымен қатар жинаққа жүргізілетін жалғыз операция көмегімен жинаққа енетін семафор мәндерін өзгертуге болады, яғни бір уақытта бірнеше P немесе V операцияларын қолдануға болады, немесе осы операцияларды біріктіруге болады.

Пайда болған семафорлар жинағын ашу немесе құру үшін UNIX –те semget() функциясы қолданылады:

```
#include <sys/sem.h>
/*В Linux - #include <linux/sem.h>*/ int
semget (key_t key, int num, int flag);
```

Параметр key, семафор жинағы идентификаторын береді, егер оның орнына _PRIVATE тұрақты шамасы көрсетілген болса; семафор жинағы жинақпен және оның топтарымен құрылған тек процеспен қолданылуы мүмкін. Параметр num, жинақтағы семафор санын береді, көптеген UNIX-тәрізді ОЖ-де бұл параметр 25- тен аспауы қажет.

Параметр flag, келесі мәндерге ие болуы мүмкін:

- 0 — егер key идентификаторлы семафор жинағы болса, онда функция оның дескрипторын қайтарады;
- IPC_CREAT — key идентификаторлы жаңа жинағын құрады.

Егер IPC_CREAT тұрақты шамасы, логикалық қосылу операциясының IPC_EXCL тұрақты шамасымен біріккен болса, онда бұл жағдайда егер, мұндай идентификаторлы семафорлардың болуында, функция -1 қайтарады. Семафор жинағына қол жеткізу құқығын беру үшін олардың сегіздік ұсынуларымен қалған тұрақты шамаларын біріктіру қажет.

Семафор жинағына арналған R қол жеткізу құқығы, оның мәнін оқуға, w қолжетімділік құқығы — семафор мәнін өзгертуге, ал x қолжетімділік құқығы— семафор жинағының параметрін өзгертуге мүмкіндік береді.

Semget() функциясының типтік шақыртуы мынадай көрінеді: `int sem_descr;`

```
sem_descr = semget(ftok("A",1), 1, IPC_CREAT | IPC_EXCL
| 0644);
```

Семафорды құрғаннан кейін оның мәні ешқандай инизацияланбайды, сондықтан оны пайдаланбас бұрын жинақтың әр семафорына бастапқы мәнін тағайындау керек. Бұны semctl() функциясының көмегімен іске асыруға болады:

```
#include <sys/sem.h>
/*B Linux - #include <linux/sem.h>*/
int semctl (int semfd, int num, int cmd, union
semun arg);
```

Параметр semfd семафор жинағының дескрипторын береді; параметр num — жинақтағы семафор нөмірі; cmd семафорда орындалатын команданы береді. Міндетті емес төртінші параметр arg командалар параметрін береді.

Төртінші параметрдің құрылымы келесі:

```
union semun {
int val; /* SETVAL командасымен қолданылады*/ struct
semid_ds *buf; /*
                        IPC_SET және IPC_STAT
                        командасымен қолданылады */
ushort *array; /* SETALL командасымен қолданылады.
```

Семафорда жүретін негізгі командалар, мыналар:

- IPC_RMID — семафор жинағын өшіреді;
- IPC_SET — семафор жинағына arg.buf құрылымынан мәндерімен параметр орнатады;
- IPC_GET — arg.buf құрылымында семафор жинағының параметрлер мәндерін оқиды;
- GETALL — arg.array алабында семафордың барлық мәндерін оқиды;
- SETALL — arg.array алабынан барлық семафордың мәндерін оқиды;
- GETVAL — num нөмірімен семафордың мәнін қайтарады. Аргумент arg қолданылмайды;
- SETVAL — num нөмірімен семафор мәндерін arg.val орнатады.

Сәтсіз орындалуы кезінде функция -1 қайтарады.

Барлық семафор мәндерін орнату үшін SETALL командасын немесе SETVAL командасының циклдік шақыртуын қолдануға болады:

```
#include <unistd.h>
#include <sys/sem.h> /* B Linux - #include <linux/sem.h>
*/

typedef int semnum; int main()
{
int i;
int sem_descr; union semnum arg; arg.val = 4;

sem_descr = semget(ftok("A"), 3,
IPC_CREAT | IPC_EXCL | 0644); for (i=0;i<3;i++)
{
semctl(sem_descr, i, SETVAL, arg);
}
}
```

Семафорларды инициализациялау және құру операцияларын бөлу, семафор жинағы құрылған, бірақ инициализацияланбаған жағдайға әкеп соқтыруы мүмкін, ал басқа процес оны қолдануға тырысады. Мұндай жағдайдың алдын алу үшін семафор құрылған соң шамалы уақыттан кейін оны еріксіз қолдануды бастауға болады. Мұндай жағдайдан қашудың тиімді, бірақ күрделі тәсілдері [18] қарастырылған.

Қазіргі уақытта жүйеде құрылған барлық семафорды қарау үшін ipcs командасы қолданылады. Оны командалық жолдан -s параметрмен іске асыруға болады.

```
$ ipcs -s

------ Message Queues --------
key        semid  owner    perms  nsems
0x00a1684b 0      admin    644    1
0xffffffff 18     nikita   644    1
0xffffffff 24     sergey   644    1
```

Семафор параметрлері kernel.sem. құрауыш параметрлерімен анықталады. Бұл параметр бірнеше параметршілерге ие: semmni, semmsl, semmns және semopm. Semmni параметрі семафор алабының максималды санын, semmsl — алаптағы семафордың максималды санын анықтайды, semmns параметрі жүйедегі семафордың максималды санын береді және semmsl және semmni туындысы сияқты есептеледі. Параметр semopm семафорда жүргізілетін, бір рет орындалатын операциялардың максималды санын береді.

Бұл параметрлер хабарламалардың кезегі жүйесінің параметрлері сияқты оперативті өзгертуге болады.

```
echo 250 256000 32 1024 > /proc/sys/kernel/sem
```

Бірінші сан semmsl параметрінің, екінші —semmns параметрінің, үшінші —seopm параметрі мен төртінші —semni параметрінің мәнін береді.

Бұл жүйедегі өзгерістерді сақтау үшін және қайта жүктелімнен кейін бұл файлдардың жаңа мәнін /etc/sysctl.conf (kernel.sem=250 256000 32 1024) файлға енгізу керек немесе sysctl -w kernel.sem="250 256000 32 1024" командасын пайдалану керек.

Семафор параметрінің мәні туралы ақпарат ipcs –ls командасының көмегімен алуға болады:

```
$ ipcs -ls
Semaphore Limits
max semaphores per array = 250                    //
SEMMSL
max semaphores system wide = 32000                //
SEMMNS
max ops per semop call = 32                       //
SEMOPM
semaphore max value = 327 67
```

Жинақтағы семафор мәнін басқару үшін semop() функциясы қызмет етеді:

```
#include <sys/sem.h>
/* В Linux - #include <linux/sem.h> */
int semop(int semfd, struct sembuf* opPtr, int
len);
```

Параметр semfd, семафор жинағының дескрипторын береді, opPtr —әр элементі жинақтағы семафорға жүргізетін бір операцияны беретін, алаптарды көрсетеді; параметр len, жинақтың қанша элементті құрайтынын анықтайды.

OpPtr алабының элементтері келесі түрмен анықталады:

```
struct sembuf {
short sem_num; /* жинақтағы семафор нөмірі*/
short sem_op; /* семафорға жүргізілетін операция
*/ short sem_flg; /* операция жалауы */
}
```

Sem_op семорына жүргізілетін операция келесі мәндерді қабылдайды:

- 0 — семафор мәнін оқу; егер ол нөлге тең болса, онда процесс орындалуы, семафор мәні оң болғанға дейін тоқтатыла тұрады;
- Оң сан — семафор мәнін берілген бірлікке дейін арттырады;
- Теріс сан — семафор мәнін берілген бірлікке дейін азайтады;

Егер семафор мәні теріс болса, онда ядро сол сияқты процестің орындалуын тоқтатады. Процестің орындалуы, семафор мәні теріс болмағанда ғана жалғасады.

Егер sem_flg жалаушасы ретінде IPC_NOWAIT тұрақты шамасын берсе, процесс тоқтатылуы болмайды. Егер жалауша ретінде SEM_UNDO тұрақты шамасын көрсетсе, ядро семафор өзгерісін аңдиды. Егер процесс, семафор мәнін нөлге дейін немесе теріс санға дейін азайтса, онда аяқталады, семафорды ашу үшін процестерді «мәңгі» күтуді шақырмас үшін онымен жасалған өзгерістер болмайды.

UNIX әр түрлі нұсқаларында sembuf құрылымында жолақшалардың бақылау реті айырылады, сондай-ақ басқа да жолақшалар қатысуы мүмкін, сондықтан мәндерді ретімен емес, ал жолақшалардың атауы көмегімен тағайындалады.

Келесі үлгіде екі процесс семафорды экранға мәтінді шығару процесін диспетчерлендіру үшін қолданады. Екі процесс экранға мәтіндік жолдарды кезекпен шығарады, сонымен қатар sleep() функциясының көмегімен «шығару» топ-процесі 4 с, аталық — 1 с алады. Шығарудың кезектігін сақтау үшін ресурсты сақтау (берілген

жағдайда— терминал), процестің әрі V операциясын орындайды, ал босағаннан соң —P операциясы орындалады. Семафорға жүргізілген барлық операциялар төменде келтіретін үлгіде *op* алабында сақталады, сонымен қатар op[0] мәні, семафор мәнін бірлікке азайтады және V операциясына сәйкес келетін op[1] семафор мәнін нөлге тексереді және мысалда қолданылмайды, op[1] семафор мәнін бірлікке арттырады және P операциясына сәйкес келед

```
#include <sys/sem.h>
/* B Linux - #include <linux/sem.h> */
#include <time.h>
#include <unistd.h>
#include <stdio.h>
#include <signal.h>

struct sembuf *op[3]; /* семафорға жүргізілетін
операциялар:
                         op[0] - мәнін азайтады
                         op[1] - тексереді
                         op[2] - мәнін арттырады */

int main(void)
{
int fd, i,j, *status; pid_t pid;

setvbuf(stdout,(char*)NULL,_IONBF,0); for (i=0;i<3;i++)
{
/* операцияда жадыны белгілейді*/
p[i]=(struct sembuf*)malloc(sizeof(struct sembuf));
}

for (i=-1;i<2;i++)
{ /* операцияларды толтыру */ op[i+1]->sem_num = 0;
op[i+1]->sem_op = i; op[i+1]->sem_flg = 0;
}
/* нәтижесінде  {0,-1,0},{0,0,0},{0,1,0} */ witch (pid =
fork() )
{
case -1 аламыз:
perror("Bad fork\n");
exit(1);
break;

case 0:
/* child body */ sleep(1);
```

```
/*  семафорды ашу */
fd = semget(ftok("A",1), 1, 0);
for (i=0;i<10;i++)
{
semop(fd, op[0], 1); /* V семафор */ printf("c%d ",i);
/* процесс ұзындығын имитациялау (4 сек) */ sleep(4);
semop(fd, op[2], 1); /* P семафорда */
}
break;

default:
/* parent body */
/* семафорды құру */
fd=semget(ftok("A",1), 1, IPC_CREAT | 0644);
/* семафордың бастапқы мәнін орнату в 1 */ semctl(fd, 0,
SETVAL, 1); for (i=0;i<10;i++)
{
semop(fd, op[0], 1); /* V семафор */ printf("p%d ",i);
/* қысқа процесті имитациялау (1 сек) */ sleep(1);
semop(fd, op[2], 1); /* P семафор */
}
wait(&status); /* ұрақтың аяқталуын күту */ semctl(fd,
0, IPC_RMID); /* семафорды жою */ break;
}
printf("\n");
/* операция жадын босату */ for (i=0;i<3;i++)
free(op[i]); return 0;

}
```

Бұл бағдарламаны іске қосудың нәтижесінде экранға келесі реттілікті шығарады, онда процестің біріктірілуінің орындалуын көрсетеді:

```
c0 p0 cl pi c2 p2 c3 p3 c4 p4 c5 p5 c6 p6 c7 p7 c8 p8 c9
p9
```

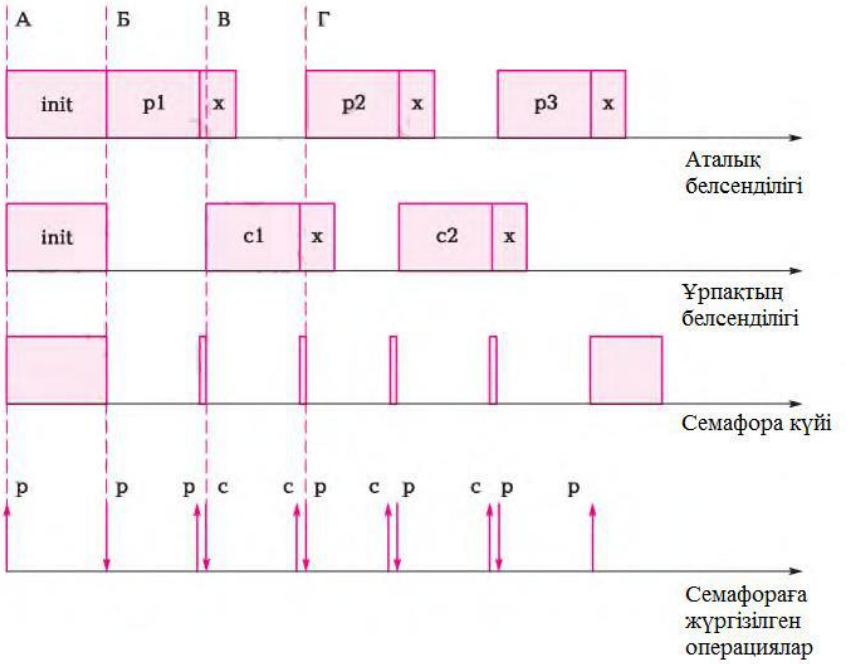
яғни аталық-процесс, семафор ашылғанға дейін шығарылуы 1 с алады,

3 с күтеді, баяу топ-процеспен қондырылатын, шығарылым 4 с алады. Ең баяу процесс, ресурс босағанға дейін 1 с күтеді.

Егер жоғары келтірілген бағдарламалар semop() функциясы шақыртуларын кетірсе, онда бағдарламаның шығуы мынадай болады:

```
c0 p0 pl p2 p3 cl p4 p5 p6 p7 c2 p8 p9 c3 c4 c5 c6 c7 c8
c9
```

Екі нұсқаны салыстырмасынан синхрондаудің жоқтығының айырмашылықтары анық көрініп тұр— бұл уақытта топ-процесі, бір «cX» реттілігін шығарып үлгереді, аталық төрт «pX» тізбектілігін шығарады.



10.7- сурет. Процестерді синхрондау үшін семафорды қолданудың уақытша диаграммасы

Семаформен жұмыс процесін уақыт сызығында сипаттасақ 10.7-суретте көрсетілген бейнені алуға болады. Суретте бірнеше уақыт сызықтары бейнеленген, әрі процесс күйінің немесе семафордың өзгеруіне сәйкес келеді.

Уақыт сызығында тіктөртбұрыштар процестің белсенділігіне (алғашқы екі сызық) және семафор мәнінің оң (ашық) кезеңіне (үшінші сызық) сәйкес келеді. Уақыттың төменгі сызығы семафорға жүргізілетін операцияны көрсетеді; төменге бағытталған нұсқармен семафордың мәні азаюын және жоғарыға бағытталған нұсқалар— мәндердің артуын белгілейді. Нұсқалардың жанында операцияны орындаған процесс көрсетіледі. «P» әрпімен аталық - процесс, «c» әрпімен — топ-процесс бейнеленген.

Екі процесс бір уақытта басталады және алдымен жеке ақпараттарын инициализациялайды, сонымен бірге аталық семафорды ашады (A кесіндісі). Инициализация аяқталған соң, аталық-процесс семафорды жабады, осымен сыни секцияны бастайды (графиктегі Б және p1 кесіндісі).

Топ-процесс семафорды жабу кезінде тоқтайды. Өзінің сыни секциясы аяқталған соң аталық семафорды ашады және сыни секцияға кірмейтін бағдарламалық кодты орындауды жалғастырады (графиктегі x). Осы сәтте топ-процесс белсенді болады, семафордың мәнін азайтады және критикалық секцияда өзінің орындалуын бастайды (графиктегі В және c1 кесіндісі).

Аталық критикалық секцияға кірмейтін код аяқталған соң, семафорды жабу кезінде өзінің орындалуын тоқтата тұрады және семафор мәнін азайта алған соң өзінің жұмысын бастайды (Г кесіндісі). Одан кейін процесс қайталанады.

10.6.

WINDOWS ПРОЦЕСТЕРІ ЖӘНЕ ПРОЦЕСАРАЛЫҚ ӨЗАРА ӘРЕКЕТТЕСУ

Windows тобының операциялық жүйелері, әсіресе Windows NT ядросы негізінде жүйелер (Windows 2000, Windows XP, Windows Vista), процестерді синхрондау үшін құралдардың кең жинағын

ұсынады және процесаралық өзара әрекеттесуді ұйымдастырады. Осы тарауда біз Windows тобының операциялық жүйелеріндегі өзара әрекеттесуді ұйымдастырудың кейбір тәсілдерін қарастырамыз.

10.6.1. Процесстер мен ағындар

Басқа операциялық жүйелердегі сияқты *процестер* (process) пайдаланушыға міндеттерді шешуге мүмкіндік беретін, негізгі обьектілер болып табылады. Әр процесс сәйкес келетін қосымшаларды орындауға қажетті ресурстарды ұсынады. Әр процесс өзіне орайластырылған виртуалды мекенжай кеңістігіне; орындалатын код; ашық жүйелік обьектілермен байланысты дескрипторлар; қауіпсіз мәнмәтіні; процестің бірегей идентификаторы; айнымалы шеңберлер; бастапқылық класы; процеске қолжетімді виртуалды жадының минималды және максималды өлшемі; ең кемі басқарудың бір ағымына ие.

Әр процесс старт кезінде бір-жалғыз *басқару ағымын* (thread) іске асырады, оны *алғашқы ағым* (primary thread) деп атайды. Бірақ әр ағын жаңа ағым құра алады. Бұл мағынада процесс негізгі есептеуші жұмысын атқаратын көптеген ағымдардың капсулданатын контейнерін ұсынады.

Процестің барлық ағымдары өздерінің арасында виртуалды мекенжай кеңістігін және жүйелік ресурстарды бөліседі. Одан басқа, әр басқару ағымы ерекше жағдайлардың жеке өңдеушілеріне, басымдылыққа, ағымның жергілікті жадына, ағымның бірегей идентификаторына және ағым қазіргі мәнмәтін туралы деректеріне ие. *Ағым мәнмәтіні* (thread context) процессор тіркелімінің қазіргі мәндерін құрайды; ядро шақыртулар қамшысы, ағым шеңберінің блогы, ағым қамшысының өлшемі туралы ақпараттар құраушы және аталық процестің мекенжай кеңістігіндегі пайдаланушы қамшысы. Одан басқа, ағымдар құқықты аталық процестен мұраға алмай шеттен алып пайдаланған жағдайда қауіпсіздіктің жеке мәнмәтініне ие болады.

Windows NT (Windows NT 3.x — 4.0, Windows 2000, Windows XP, Windows Vista және Windows тобының барлық серверлік операциялық жүйелері) ядросына және Windows 9x (Windows 95, Windows 98 и Windows ME) негізделген операциялық жүйелер *ығыстырылған көпміндеттілікті (*preemptive multitasking) қолдайды. Ол бірнеше процесте бірнеше ағымдардың орындалуының әсерін құруға мүмкіндік береді. Windows-ң көптеген алғашқы жүйелерінде (мысалы, Windows 3.x құрамасы) *ығыстырылмайтын көпміндеттілігіне* (nonpreemptive multitasking) негізделген, ағымдардың бір уақытта орындауының қарапайым моделін ұстанған болатын.

Ығыстырылатын көпміндетті операциялық жүйе кезінде, ал нақтырақ, *ығыстырылатын жобалаушы* (preemptive scheduler) деп

аталатын арнайы жүйелік процесс уақыт шегіне байланысты оны кідіртетін жағдайға ауыстыра отырып, ағындағы процесті уақытша тоқтатады. Одан кейін жобалаушы, оның басымдылығына байланысты, ертерек кідірткен процестерді оятады және осы процесс үшін процессорлық уақыттың квантын белгілейді. Осы механизм «мәнмәтінді ауыстыру» (context switching) атауына ие. Ығыстырылмайтын көпміндеттілік кезінде жадыда бір уақытта бірнеше процесс қатысуы мүмкін, бірақ процессорлік уақыт, процестің өзі немесе пайдаланушы процессорды босатқанға дейін, негізгі процеске белгіленеді.

Көп процессорлы жүйелерде, операциялық жүйелер Windows NT ядросының (немесе ядролар) негізінде жүйеде орнатылған қанша процесс болса, сонша ағымдардың бір уақытта орындалуына мүмкіндік береді. Бұл жағдайда, ұқсастыру емес, шын көпміндеттілік пайда болады.

Пайда болған жаңа процестер үшін CreateProcess() функциясы қолданылады:

```
include <windows.h>
BOOL WINAPI CreateProcess(
   LPCTSTR lpApplicationName,
   LPTSTR lpCommandLine,
   LPSECURITY_ATTRIBUTES lpProcessAttributes,
   LPSECURITY_ATTRIBUTES lpThreadAttributes,
   BOOL bInheritHandles,
   DWORD dwCreationFlags,
   LPVOID lpEnvironment,
   LPCTSTR lpCurrentDirectory,
   LPSTARTUPINFO lpStartupInfo,
   LPPROCESS_INFORMATION lpProcessInformation);
```

CreateProcess() функциясын шақырған процесс, аталық-процесс деп аталады, ал бұл функцияны шақырудың нәтижесінде пайда болған процесс процесс-тобы деп аталады. Процесс-тобы, оны туындатқан процеске толық тәуелді емес. Бірақ аталық процесс туындаған процесті бақылауға және онымен байланысты кейбір оқиғаларды барлауға мүмкіндік алады.

LpApplicationName параметрі орындалуға міндетті бағдарламаның атауын береді. Егер осы параметр NULL тең болса, онда іске

асырылатын бағдарламаның атауы lpCommandLine параметрінде берілуі мүмкін. Сондай-ақ осы параметрде, іске асырылатын бағдарламаға жіберілетін параметрді де береді.

Егер lpCommandLine параметр NULL мәніне ие болса, онда іске асырылатын бағдарлама lpApplicationName параметрінен алынады. Егер екі жол да NULL тең болмаса, онда lpApplicationName параметрі іске асырылатын бағдарламаны береді, ал параметрінде lpCommandLine осы бағдарламаға арналған аралықтармен бөлінген тізімді жібереді.

LpProcessAttributes параметрі құрылатын процеске, әдепкі құқыққа қарағанда қолжетімділік құқығын береді. Одан басқа, параметр көрсететін құрылымның бір элементі, CreateProcess() функциясының шақыртуы нәтижесінде құрылған процесс дескрипторын көрсетуге қолданылады, ол процесс-тобын мұраға алуға болады.

LpThreadAttributes параметрі, lpProcessAttributes параметріндей қолданылады. Бірақ егер lpProcessAttributes параметрі құрылған процестің параметрін өзгертуге арналған, онда lpThreadAttributes параметрінде жіберілетін ақпараттар құрылатын процестің алғашқы ағымдарының параметрлерін өзгертуге қолданылады.

BInheritHandles параметрі еншілес процесс аталық-процестен мұраға қалатын дескрипторды мұраға ала ма (TRUE) және жоқ па (FALSE) соны көрсетеді. Сонымен қатар, ашылған файлдың дескрипторы ғана емес, каналдар және басқа жүйелік ресурстардың құрылған процестің дескрипторын да мұраланады. Мұраланған дескрипторлар, қолжеткізу құқығындағы мәндерге ие болады. Айта кететін жайт, барлық дескриптор емес, мұралануға белгіленгендер ғана мұраланады. Дескриптордың бұл қасиеттері, процесс аралық өзара әрекеттесуді ұйымдастыру кезінде маңызды.

DwCreationFlags параметрі құрылатын процестің басымдылық класын беру үшін сондай-ақ процестің қасиеттерін басқару үшін де қолданылады. Мысалы, егер аталық-процесс және құрылатын процесс, консолды қосымша болып табылады және параметрде CREATE_NEW_CONSOLE мәні беріледі, онда құрылатын процесс өзінің консолды жеке терезесіне ие болады. Бұл көрсетпегенше жаңа терезе ашылмайды, ол құрылған процесс аталық-процестің консолды терезесін мұраға алады.

LpEnvironment параметрі, құрылатын процестің айнымалы шеңберін өзгерту үшін қолданылады және жаңа процесс үшін шеңбер блогына нұсқаушыны құрайды. Бұл блок нөлмен аяқталады және мына жолдардан тұрады:

```
name=value\0
```

егер берілген параметр NULL мәніне ие болса, шеңбер аталық-процестен мұраланады.

LpCurrentDirectory параметрі, құрылатын процесс үшін каталогты және ағымдағы дискіні беру үшін қолданылады. Егер параметр NULL тең болса, онда ағымдағы дискі және каталог аталық-процестен мұраланады.

LpStartupInfo параметрі, ол да құрылатын процестің ерекшеліктерін өзгертуге арналған. Берілген параметр құрылатын процес терезесінің бастапқы координаталарын беруге мүмкіндік береді, терезені көрінетін немесе жасыру керектігін анықтайды, сондай-ақ, аталық-процеспен, процесс-тобымен консолға шығарылатын ақпараттарды алуға немесе бұл ақпараттарды файлға не құрылғыға қайта бағыттауға арналған стандартты құрылғылардың дескрипторын қайта анықтауға мүмкіндік береді. Осы параметр құрылатын процестің басқа да ерекшеліктерін өзгертуге мүмкіндік береді.

Параметр lpProcessInformation, қайтарылушы болып табылады. Бұл параметрде құрылған процестің және оның бастапқы ағынының идентификаторы мен дескрипторын қайтарады. Берілген функция 0 қайтарады, егер белгілі бір себеппен жаңа процесс құралмаса немесе мәні 0-ден айырмашылықта болса, онда процесс сәтті құрылған.

Жаңа процесті құруда жүйеден бас тарту себептерінен, виртуалды жадының таусылуы, іске асырылатын бағдарламаның бағдарлама немесе сценарий болмауы, процесс кестелерінің толып қалуы мүмкін. Одан басқа, жаңа процестің туылуына, бір процесс үшін дескриптор саны шегіне жауап беретін, реестрдің HKLM\ Software\Microsoft\Windows NT\CurrentVersion\Windows\ UserProcessHandleQuota параметрі әсер етеді. Әдеттегідей Windows XP және Vista жүйелеріндегі осы параметр үшін 10 000 мәні орнатылған, бірақ жүйенің әкімшілігі бұл мәнді өзгерте алады және осылайша қосылатын бағдарламаның саны шектелуін өзгерте алады.

Сондай ақ қосылатын процесс үшін қорытынды команданың ұзындығына жүйелік шектеу қойылады. Ол ұзындығында 32 767 символдан аспауы қажет.

Жаңа ағымдарды құру үшін CreateThread() функциясы қолданылады.

```
include <windows.h>
HANDLE WINAPI CreateThread(
```

```
LPSECURITY_ATTRIBUTES lpThreadAttributes,
SIZE T dwStackSize,
LPTHREAD_START_ROUTINE lpStartAddress,
LPVOID lpParameter,
DWORD dwCreationFlags,
LPDWORD lpThreadId);
```

LpThreadAttributes параметрі, әдеттегідей құқықтардың үздігінен, қолжетімділік құқығында құрылатын процестер міндеттері үшін қолданылады. Сондай-ақ осы параметр, құрылған ағымның дескрипторының процесс тобынан мұралануы болу мүмкіндігін көрсету үшін қолданылады. Егер де параметр NULL мәніне ие болса, ағым әдеттегідей құқығын алады, ал ағымның дескрипторы тұқыммен мұралануы мүмкін емес.

DwStackSize параметрі, байттағы ағымның бастапқы қамшы өлшемін беруі үшін арналған. Егер де параметр 0 мәнін берсе, онда қамшы өлшемі, қосымшалар үшін анықталған қамшы өлшеміне сәйкес келеді.

LpStartAddress параметрінде ағыммен орындалатын, қосымшаның жергілікті функциясының мекенжайын береді.

LpParameter параметрі, ағымның негізгі функциясына қажетті, құрылған ағымдарға кіретін мәндерді жіберуі үшін арналған.

DwCreationFlags параметрі ағымның құрылуын басқарады. Мысалы, егер осы параметр CREATE_SUSPENDED мәніне ие болса, онда құрылған ағым, басқа басты ағым оны жаңадан іске қосқанша, кідіре тұрады.

Қайтарылатын lpThreadId параметрі құрылған ағым идентификаторын алу үшін қолданылады. Егер осы параметр NULL тең болса, онда құрылған ағымның идентификаторы қайтарылмайды.

Егер функцияның шақыртуы жаңа ағымның құрылуымен аяқталса, функция құрылған ағымның дескрипторын қайтарады. Қарсы жағдайда NULL мәнін қайтарады.

Жаңа ағым құруда сәтсіз талпыныстардың себебі, жүйе ресурстарының немесе дескрипторда лимиттің таусылуы болып табылады. Жаңа ағымның құрылуында сәтсіз талпыныстары болып жиірек виртуалды жадының таусылуы табылады, егер жүйеде ағымдардың саны көп болса.

Ағымдағы процестің дескриптор мен идентификаторын қабылдау үшін GetCurrentProcess() және GetCurrent- ProcessId() функциялары қолданылады, сәйкесінше:

```
include <windows.h>
HANDLE WINAPI GetCurrentProcess(void);
DWORD WINAPI GetCurrentProcessId(void);
```

Үқсас функциялар бар және қазіргі ағымның дескриптор мен идентификаторды алу үшін:

```
include <windows.h>
HANDLE WINAPI GetCurrentThread(void);
DWORD WINAPI GetCurrentThreadId(void);
```

Жаңа ағымдар мен процесті туындататын бағдарламаның үлгілерін қарастырамыз:

ProcessAndThread.c:

```
#include <windows.h>
#include <stdio.h>
#include <tchar.h>
// құрылатын ағымға берілетін ақпараттар
typedef struct _Data {
   TCHAR cmd[20];
   int parentThreadID;
} TDATA, *PTDATA;

DWORD WINAPI ThreadProc(LPVOID lpParam);

void ErrorReport(LPTSTRlpszFunction);

int main()
{
      PTDATA pData;
      DWORD dwThreadId;
      HANDLE hThread;
// құрылатын ағымға берілетін ақпараттар жадын
белгілейміз
pData = (PTDATA)HeapAlloc(GetProcessHeap(),
HEAP_ZERO_MEMORY, sizeof(TDATA));
```

```
    if(pData == NULL)
    {
    ErrorReport(TEXT("HeapAlloc()"));
    return(1);
    }
    // ақпараттар құрылымын толтырамыз: шығаратын команда
    // айнымалы шеңберлер және қазіргі ағымның
    идентификаторы
    // ағымда
    _tcscpy(pData->cmd, TEXT("cmd /c set ComSpec"));
    pData->parentThreadID = GetCurrentThreadId();
    // жаңа ағым құрамыз
                    hThread = CreateThread(
      NULL,         // әдеттегідей қауіпсіздік атрибуттары
      0,            // әдеттегідей ағым қамшысының өлшемі
       ThreadProc,// pData ағымының негізгі функциялары,
              //ағымға арналған ақпарат
       0,     // әдеттегідей ағым құрудың жалауы
       &dwThreadId); // идентификаторын құрайды
                     // құрылған ағымның
    // ағымның сәтті құрылғанын тексереміз
    if (hThread == NULL)
    {
    ErrorReport(TEXT("CreateThread()"));
    return(1);
    }
    // тайм-аутсыз ағымның аяқталуын күтеміз
    WaitForSingleObject(hThread, INFINITE);
    // CloseHandle(hThread) ағымының дескрипторын
    жабамыз;
    return(0);
}
// DWORD WINAPI ThreadProc(LPVOID lpParam)құрылатын
ағымның басты функциясы
{
    STARTUPINFO si;
    PROCESS_INFORMATION pi;
    PTDATA pData;
    // pData = (PTDATA)lpParam құрылатын ағымына
    жіберілетін ақпаратты аламыз;
    // аталық-процесс туралы ақпараттың консолын аламыз
```

```
   // туындайтын процесс туралы ақпараттың консолын
   аламыз
   // құрылған ағым
   _tprintf (TEXT("New thread is created by thread with
   Id %d and command to execute is \"%s\"\n"),
   pData->parentThreadID, pData->cmd);
   ZeroMemory(&si, sizeof(si));
   si.cb = sizeof(si);
   ZeroMemory(&pi, sizeof(pi));
   // процесс-тобын қосамыз
   if(!CreateProcess(NULL,// бірінші параметрді аламыз
                           //
      pData->cmd,          // Командалық жол

      NULL,         //  мұраланатын құрылатын
                    //процесс  дескрипторы
                    // Дескриптор первичного
                    //
      FALSE,        // мұраланбайтын ағым,
                    // Процесс дескрипторы аталық
                    // процестен мұраланбайды
      0,            //
      NULL          // әдеттегідей процесс құру жалауы,
      NULL          // аталық - процесінен
                    // шеңбер мұраланады
                    // қазіргі каталог
                    //аталық процестен мұраланады
      &si,          // бастапқы қондырғыға нұсқау
                    // процестер үшін
      &pi)          // процестен және алғашқы ағым
      дескрипторын аламыз
)
{
   ErrorReport(TEXT("CreateProcess()"));
   return(1);
}

   // процесс тобы орындалуының аяқталуын күтеміз //
   тайм-аутсыз
        WaitForSingleObject(pi.hProcess, INFINITE);

   // процесс дескрипторын және оның алғашқы ағымын
```

```
   жабамыз //
   CloseHandle(pi.hProcess);
   CloseHandle(pi.hThread);

   // ағым параметрлерге белгіленген жадыны босатамыз

   HeapFree(GetProcessHeap(), 0, pData); return(0);
}

void ErrorReport(LPTSTR lpszFunction)
{
   LPVOID lpMsgBuf;
   LPVOID lpDisplayBuf;
   DWORD dw = GetLastError();

   FormatMessage(
      FORMAT_MESSAGE_ALLOCATE_BUFFER |
      FORMAT_MESSAGE_FROM_SYSTEM,
      NULL,
      dw,
      MAKELANGID(LANG_NEUTRAL, SUBLANG_DEFAULT), (LPTSTR)
      &lpMsgBuf,
      0, NULL ) ;

   lpDisplayBuf   =   (LPVOID)LocalAlloc(LMEM_ZEROINIT,
      (lstrlen((LPCTSTR)lpMsgBuf)+lstrlen((LPCTSTR)
      lpszFunction)+40)*sizeof(TCHAR));
   _stprintf((LPTSTR)lpDisplayBuf,
   TEXT("%s failed with error %d: %s"), lpszFunction,
      dw, lpMsgBuf);
   MessageBox(NULL, (LPCTSTR)lpDisplayBuf,
      TEXT("Error"), MB_OK);

   LocalFree(lpMsgBuf);
   LocalFree(lpDisplayBuf);
}
```

Берілген бағдарламамен жұмыс процесінде, екі параметрді қабылдайтын: командадан, аталық ағымның идентификаторы мен жолдан тұратын тағы бір ағым құрылады. Өз кезегінде, құрылған ағым, өзіне берілген команданы (қарау командасының мысалында, айнымалы шеңбердің мәндері ComSpec — «cmd /c set ComSpec») қолдана отырып, жаңа процесс туындатады.

Жоғарыда келтірілген бағдарлама жұмысының нәтижесінде консолға хабарлама шығады:

```
C:\>ProcessAndThread.exe
New thread is created by thread with Id 3136 and
command to execute is "cmd /c set ComSpec"
ComSpec=C:\WIN\system32\cmd.exe

C:\>
```

Берілген мысалда қарастырылған, тағы бір функция қатарын еске салған жөн. Алдымен, ол WaitForSingleObjectQ функциясы:

```
include <windows.h>
DWORD WINAPI WaitForSingleObject(
   HANDLE hHandle,
   DWORD dwMilliseconds);
```

Бұл функция қазіргі процесте жіберілген hHandle дескрипторына ұқсатылған обьект, сигналды күйге енбегенше немесе берілген dwMilliseconds күту кезеңі милисекундта өтуіне дейін блоктайды. Егер dwMilli- seconds параметрі INFINITE мәніне ие болса, онда күту кезеңі өтіп кетпейді және процесс, берілген обьектінің сигналды күйге өтуі кезінде оянады. WaitForObjects() тобының функциялары, Windows тобының операциялық жүйелерімен сүйемелденетін синхрондауші механизмдер элементінің бірі болып табылады.

Берілген функцияның күйлерін бақылай алатын обьектілер ретінде, оқиғалар, мьютекстер, процесстер, семафорлар, ағымдар және т.б. қатыса алады. Егер обьектілер, процестер немесе ағымдар бақыланатын болса, онда олар үшін сигналды күй олардың аяқталғанында түседі.

Егер функция, бақыланатын обьектілердің сигналды күйге өтуі кезінде аяқталса, онда WAIT_ OBJECT_0 мәні қайтарылады. Егер де функция күту кезеңінің өтуімен аяқталса, онда WAIT_TIMEOUT мәні қайтарылады.

Егер бір уақытта бірнеше обьектілерді бақылау қажет болса, онда WaitForMultipleObjectsQ функциясы қолданылады.

```
DWORD WINAPI WaitForMultipleObjects(
   DWORD nCount,
   const HANDLE* lpHandles,
   BOOL bWaitAll,
   DWORD dwMilliseconds);
```

Бұл функцияда nCount параметрі, обьектілер дескрипторы жалпы санын анықтайды. LpHandles параметрінде обьектілер дескрипторының алабы беріледі, олардың күйі WaitForMultipleObjects() функциясымен бақыланады. Егер bWaitAll параметрі үшін TRUE мәні қондырылса, онда функция күтілу процесін басқаруды, алаптағы барлық обьектілерді сигналды күйге ауыстыруға ауыстырғаннан кейін қайтарады. Қарсы жағдайда, функция барлығын бір уақытта емес бір обьектінің сигналды күйге ауысуын күтеді. DwMilliseconds параметрі, WaitForSingleObject() функциясы сияқты, обьектілерді сигналды күйге ауыстыруының күтілуі кезеңін беру үшін оның бітуінен, басқару процессі функциясының шақыртылуымен қайтарылады. Бұл жағдай, егерде күтілу кезеңі біткенде және обьектінің бірі де сигналдық күйге ауыспаса да жүреді. Бұл параметр сондай-ақ INFINITE мәнін қабылдай алады.

Жұмыс аяқталғаннан соң, бұл функция, егер обьектілердің сигналды күйге ауысуы күтілуінің уақыты өтсе, WAIT_TIMEOUT мәнін қайтарады. Егер функцияларды шақырту кезінде bWaitAll параметрі TRUE тең болса және барлық обьектілер күту кезеңінің өтуіне дейін ауыстырылу керек болса, функция WAIT_OBJECT_0 мәнін қайтарады. Егер де bWaitAll параметрі FALSE тең болса және қандай да бір обьект сигналды күйүге ауыстырылса, WAIT_OBJECT_0 + n мәні қайтарылады, мұнда n —сигналды күйге ауыстырылған, lpHandles алабындағы обьектінің нөмірі.

Дескрипторлары ашылатын, обьектілермен жұмысты аяқтаған соң, соңғылары жабылуы керек. Ол үшін CloseHandle() функциясы қолданылады:

```
include <windows.h>
DWORD WINAPI BOOL CloseHandle(HANDLE hObject);
```

Параметр ретінде ертерек ашылған дескриптор беріледі. Процестің дескрипторлары және ағымдар, егер процестің өздері де, ағымдар да аяқталған соң ғана жабылуы тиіс.

10.6.2. Синхрондау. Оқиғалар, семафоралар, мьютекстер

Бір уақытта жұмыс істейтін бірнеше бәсекелес процестерді

сүйемелдейтін операциялық жүйелердің көпшілігі, Windows тобының операциялық жүйелері процестер мен ағымдарды синхрондау механизмдерінің бір қатарын құрайды. Бұл процестерге, бөлінбейтін ресурстарды бір уақытта қолданудан құтылуға мүмкіндік береді, бұл процестердің бұзылуына, кейбір жағдайларда оның барлық жүйелерінің бүзылуына әкеледі.

Windows тобының операциалық жүйелері: оқиғалар, семафорлар, мьютекстер, критикалық облыстар және бәсекелесетін процестердің арасында ресурстарды бөлуді ұйымдастыратын тәсілдердің бір қатары сияқты синхрондау механизмдерін көтереді. Берілген тарауда біз оқиғалар, семафорлар және мьютекстер сияқты синхрондау механизмдерінің жұмыстарын қарастырамыз.

Оқиға (events) синхрондау механизмінің обьектісі ретінде ұсынылады, кейбір бағдарламалы басқару оқиғаларының түскенін хабарлауға арналған. Синхрондаудің екі типі бар — қолмен атқарылып тасталатын оқиғалар, және автоматты тасталатын оқиғалар.

Автоматты тасталатын оқиғалар, егер осындай оқиға сигналды күйге ауысқанда, жүйе автоматты ағымның сигналды күйге күтілуі таңдалады және оған басқаруын беруімен ерекшеленеді. Сонымен қатар, осындай оқиға сигналды емес күйге автоматты ауысады. Қолмен тасталатын оқиғалармен жұмыс кезінде, сигналды емес күйдегі оқиғаның қалпына келуі жауапкершілігін бағдарламалаушы алады. Бұл жағдайда ол, оқиғаны сигналды күйге ауыстыру қажет болғанда сайын ResetEvent() функциясын нақты шақырады.

Оқиғаның құрылуы үшін CreateEventQ функциясы қолданылады:

```
include <windows.h>
HANDLE WINAPI CreateEvent(
   LPSECURITY_ATTRIBUTES lpEventAttributes,
   BOOL bManualReset,
   BOOL bInitialState,
   LPCTSTR lpName);
```

Бірінші параметр lpEventAttributes, процесс-тобымен құрылатын оқиғаның дескрипторы бар немесе мұраға берілетін, сондай-ақ құрылатын оқиғаның қолжеткізу құқығын өзгертуге жауап береді. Егер берілген параметр NULL мәніне ие болса, онда дескриптор мұраланбайды және әдепкі жағдай бойынша қол жеткізу құқығы қондырылады.

Параметр bManualReset, оқиғаның қандай типінің құрылуы тиісті екеніне жауап береді. Егер осы параметр TRUE мәнін қабылдаса, онда

қолмен тасталатын оқиғалар құрылады. Егер де FALSE мәні берілсе, автоматикалық тасталатын оқиғалар құрылады.

Параметр bInitialState, құрылатын оқиғаның бастапқы күйін беру үшін арналған. Егер осы параметр TRUE тең болса, онда бастапқы сигналды күйдегі оқиғамен оқиғалар құрылады. Қарсы жағдайда, оқиғаның бастапқы күйі сигналды емеске қондырылады.

LpName параметр, құрылатын оқиғаның атауын беру үшін қолданылады. Егер осы параметр NULL тең болса, онда анонимді объект-оқиғалар құрылады. Егер осы параметрде аты берілсе, онда жүйе осындай аты бар оқиғаның бар-жоғын тексереді. Егер осындай оқиға болмаса, онда бастапқы қондырғылардың талаптарымен жаңа оқиға құрылады. Қарсы жағдайда жаңа оқиға құрылмайды, бұдан бұрын құрылған берілген атпен ассоциацияланған дескриптор ашылады. Құрылатын оқиғалар саны процесс үшін тек жүйелік ресурстармен және дескрипторлардың шектелген санымен лимиттеледі, олар жүйе әкімшісімен өзгертіле алады.

Жүйедегі айқын нұсқау үшін бұдан бұрын құрылған оқиғаларға дескрипторды алу қажет, ол үшін OpenEvent() функциясы қолданылады:

```
include <windows.h>
HANDLE WINAPI OpenEvent(
   DWORD dwDesiredAccess,
   BOOL bInheritHandle,
   LPCTSTR lpName);
```

dwDesiredAccess параметрі жүйеге пайдаланушының оқиғаны басқару үшін қандай рұқсат құқықтарын талап ететінін хабарлайды. Осы параметр оқиғаға толық рұқсатты алуға мүмкіндік беретін EVENT_ALL_ACCESS не EVENT_MODIFY_ STATE мәнін өзгерте алады. Бұл мән пайдаланушыға оқиғаның жай-күйін өзгерте алады және көп жағдайларда оның жұмыс істеуіне осы жеткілікті болады.

Параметр bInheritHandle, оқиғаның құрылатын дескрипторы еншілес процеске мұралана ма, жоқ па, соны анықтайды. Егер осы параметр TRUE тең болса, онда құрылатын дескриптор процесс-тобымен мұраланады, басқа жағдайда мұраланбайтын дескриптор құрылады.

Параметр lpName, бар оқиғаның атауын береді, пайдаланушы оған қолжеткізу құқығына жеткісі келеді. Осылайша, осы функцияның көмегімен тек атаулы оқиғаларға қол жеткізуге болады. Анонимді оқиғалар, оқиға құратын процестерді және ағымдарды қолдануы

мүмкін.

Кез келген тип оқиғасының сигналды күйге ауысуы SetEvent() функциясымен жүзеге асырылады.:

```
include <windows.h>
DWORD WINAPI SetEvent(HANDLE hEvent);
```

Параметрі ретінде осы функцияға, сол оқиғаның дескрипторы беріледі, ол сигналды күйге ауысуы керек.

Оқиғаның тасталуы үшін ResetEvent() функциясы қолданылады:

```
include <windows.h>
DWORD WINAPI ResetEvent(HANDLE hEvent);
```

Параметр hEvent, өз кезегінде, күйі сигналды жағдайдан, сигналдық емес қалпына келетін оқиғаның дескрипторын береді.

Процестер немесе ағымдар жұмысының аяқталуын күткен жағдайда, процесс басқа оқиғаның сигналды күйге ауысуы, жоғарыда қарастырылған WaitForSingleObjectQ және WaitForMultipleObjectsQ функцияларының көмегімен жүретінін хабарлайды.

Процестер мен ағымдардың синхрондау үшін оқиғалардың қолданылуына мысал ретінде, жоғарыдағы, әртүрлі уақыт орындалуы бар, екі процестің үйлестірілген мысалын қарастырамыз.:

```
#include <windows.h>
#include <stdio.h>
#include <tchar.h>

DWORD WINAPI ThreadProc(LPVOID lpParam);
void ErrorReport(LPTSTRlpszFunction);
int main()
{
   DWORD dwThreadId;
   HANDLE hThread, hEvent1, hEvent2;
   unsigned i;
   // автоматты тасталатын оқиға құрамыз //
   бастапқы сигналды емес оқиғамен
   if((hEvent1=CreateEvent(NULL, FALSE, FALSE,
   "Thread1")) == NULL)
   {
      ErrorReport(TEXT("CreateEvent()"));
      return(1);
   }
   // бірінші ағыммен синхрондауға тырысатын
```

```
ағым құрамыз // hThread = CreateThread(
   NULL,        //  әдепкі бойынша құқық
   0,           //  әдеттегідей қамшы өлшемі
   ThreadProc, // ағымның функциясы
   NULL,        //  функция үшін аргумент
   0,           //  әдеттегідей жалауы жоқ
   &dwThreadId); if
(hThread == NULL)
{
   ErrorReport(TEXT("CreateThread()"));
   return(1);
}
// екінші ағынмен тағы бір оқиғаның құрылуын күтеміз
// екінші оқиғаның құрылғанынан кейін//  ол алғашқы
оқиғаны сигналды күйге ауыстырады // ResetEvents()
функциясы шақырылмайды,
// себебі оқиға автоматты тасталынады
WaitForSingleObject(hEvent1, INFINITE);
// туындаған ағыммен құрылған оқиғаны ашамыз
if((hEvent2 = OpenEvent(EVENT_ALL_ACCESS, FALSE,
"Thread2")) == NULL)
{
   ErrorReport(TEXT("OpenEvent()"));
   return(1);
}
// ағым-ұрпағына басқаруды береміз // екінші оқиғаны
сигналды күйге SetEvent(hEvent2);
// алғашқы ағымның негізгі цикл for(i = 0; i < 10;
++i)
{
   // туындаған ағымның басқарылуын беруді күтеміз //
   WaitForSingleObject(hEvent1, INFINITE);
   // ақпараттарды консолға шығарамыз printf("p%d ",
   i);
   // алғашқы ағымды 1 сек блоктаймыз Sleep(1000);
   // туындаған ағымға басқаруды береміз
   SetEvent(hEvent2);
}
// туындаған ағымның аяқталуын күтеміз
WaitForSingleObject(hThread, INFINITE);
// оның  дескрипторын жабамыз CloseHandle(hThread);
// оқиғаның дескрипторын жабамыз
CloseHandle(hEvent1);
CloseHandle(hEvent2);
printf("\n");
```

```
   return(0);
}

DWORD WINAPI ThreadProc(LPVOID lpParam)
{
   HANDLE hEvent1, hEvent2;
   unsigned i;

// алғашқы ағыммен құрылған оқиғаны ашамыз if((hEvent1 =
OpenEvent(EVENT_ALL_ACCESS, FALSE, "Threadl")) == NULL)
   {
      ErrorReport(TEXT("OpenEvent()"));
      return(1);
   }
   // автоматты лақтыруды құрамыз C // сигналды емес
   алғашқы күймен ((hEvent2 =CreateEvent(NULL, FALSE,
   FALSE, "Thread2")) == NULL)
   {
      ErrorReport(TEXT("CreateEvent()"));
      return(1);
   }
   // алғашқы ағымға, қазіргі ағымда құрылған
   SetEvent(hEvent1) оқиғаны ашу үшін // береміз
   басқаруды;
   // еншілес ағымның негізгі циклі for(i = 0; i < 10;
   ++i)
   {
      // алғашқы ағымнан басқару жіберілімдерін алу
      //
      WaitForSingleObject(hEvent2, INFINITE);
      // ақпараттарды консолға шығарамыз printf("c%d
      ", i);
      // ағымды 4 сек блоктаймыз Sleep(4000);
      // алғашқы ағымның басқаруын жіберу
      SetEvent(hEvent1);
   }

   // оқиғаның дескрипторын жабамыз
   CloseHandle(hEvent1);
   CloseHandle(hEvent2);
   return(0);
}

void ErrorReport(LPTSTR lpszFunction)
{
   LPVOIDlpMsgBuf;
```

```
LPVOID lpDisplayBuf;
DWORD dw = GetLastError();

FormatMessage(
   FORMAT_MESSAGE_ALLOCATE_BUFFER |
   FORMAT_MESSAGE_FROM_SYSTEM,
   NULL,
   dw,
   MAKELANGID(LANG_NEUTRAL, SUBLANG_DEFAULT), (LPTSTR)
      &lpMsgBuf,
      0, NULL ) ;

lpDisplayBuf = (LPVOID)LocalAlloc(LMEM_ZEROINIT,
(lstrlen((LPCTSTR)lpMsgBuf)+lstrlen((LPCTSTR)
lpszFunction)+40)*sizeof(TCHAR));
_stprintf((LPTSTR)lpDisplayBuf,
   TEXT("%s failed with error %d: %s"), lpszFunction,
   dw, lpMsgBuf);
MessageBox(NULL, (LPCTSTR)lpDisplayBuf,
   TEXT("Error"), MB_OK);

LocalFree(lpMsgBuf);
LocalFree(lpDisplayBuf);
}
```

Осы бағдарламаның жұмысы нәтижесінде консолға келесі түрдің жолы шығады:

```
c0 p0 c1 p1 c2 p2 c3 p3 c4 p4 c5 p5 c6 p6 c7 p7 c8 p8 c9
p9
```

Ағымдарды блоктаудың әр түрлі уақыттарына қарамастан, олардың шығарылуы қатаң кезектеседі. Егер де жоғарыда келтірілген бағдарламаларда циклдің ішінде SetEvent() және WaitForSmgleObjectQ функцияларының шақыртуларын және алғашқы ағым мен ағым-тобын түсіндіре кетсе, онда бағдарлама жұмысының нәтижесі келесі түрге ие болады:

```
p0 c0 p1 p2 p3 p4 c1 p5 p6 p7 p8 c2 p9 c3 c4 c5 c6 c7 c8
c9
```

бірінші ағымның ағым-тобын еш синхрондамағаны көрініп тұр және еншілес ағымға қарағанда, өзінің негізгі жұмысын ертерек аяқтайды.

Екінші механизм Windowsti синхрондау үшін қолданылады, Linux

— семафор синхрондағыш құрылғылармен танысқанда кездестіргенбіз. Windows семафорлары ішкі есептегіші бар синхрондаушы обьектілерін ұсынады, оның мәні нөлден берілген максималды мәнге дейінгі диапозонда орналасуы мүмкін. Бұл есептегіштің мәні, ағымның біріне (WaitForSingleObjectQ и WaitForMultipleObjects()) күту функциясының басқарылуы қайтарылған сайын азаяды, ол ағым семафорды босатқанда артады. Семафор, есептегіштің мәні 0-ге тең болмағанда сигналды күйге түседі, ал есептегіш мәні 0-ге тең болғанда, сигналдық емес болады.

Семафорды құру үшін CreateSemaphore() функциясы қолданылады:

```
#include <windows.h>
HANDLE WINAPI CreateSemaphore(
   LPSECURITY_ATTRIBUTES lpSemaphoreAttributes,
   LONG lInitialCount,
   LONG lMaximumCount,
   LPCTSTR lpName);
```

LpSemaphoreAttributes параметрі процесс тобының құрылатын семафор дескрипторын мұраланатынын не мұраланбайтынына жауап береді, сондай-ақ оның қолжеткізу құқығын өзгертеді. Егер осы параметр NULL мәніне ие болса, дескриптор топпен мұраланбайды және әдеттегідей қолжеткізу құқығын орнатады.

LInitialCount параметрінде семафордың ішкі есептегішінің бастапқы мәндері беріледі. Есептегіштің максималды қолжетімді мәні lMaximumCount параметрінде беріледі.

Параметр lpName, құрылатын семафордың атауын беруге қолданылады. Егер осы параметр NULL тең болса, анонимді семафор құрылады. Егер де Windows жүйелік обьектілерін құру кезінде қолданылған атауы берілетін болса, оқиғалар, семафорлар, мьютекстер, онда семафор құрылмайды және функция қате аяқталады.

Егер функция сәтті аяқталса және семафор құрылса, онда CreateSemaphore() функциясы құрылған семафордың дескрипторын қайтарады. Басқа жағдайда функция NULL мәнін қайтарады. Құрылған семафордың саны, оқиғалар саны сияқты, жүйелік ресурспен және бір ғана процесті құратын, дескриптор саны лимитімен шектеледі.

Алдымен құрылған семафордың дескрипторын алу қажеттілігін жүйеде көрсету үшін OpenSemaphore() функция қолданылады:

```
include <windows.h>
HANDLE WINAPI OpenSemaphore(
   DWORD dwDesiredAccess,
   BOOL bInheritHandle,
```

```
   LPCTSTR lpName);
```

Параметр dwDesiredAccess, семафорды басқару үшін пайдаланушы қандай қол жеткізу құқықтарын талап етіп, жүйеге хабарлайды. Бұл параметр обьектіге толық қолжеткізуге мүмкіндік беретін SEMAPHORE_ALL_ACCESS немесе SEMAPHORE_ MODIFY_STATE мәнін қабылдайды.

Параметр bInheritHandle, семафордың құрылатын дескрипторы, еншілес процесті мұрағаттайтынын анықтайды. Егер осы параметр TRUE мәніне ие болса, онда құрылатын дескриптор процесс-тобымен мұрағатталады керісінше мұрағаттанбайтын дескриптор құрылады.

Параметр lpName қолжетімділікті алуы қажет, алдыңғы құрылған семафордың атауын береді. Осы функцияның көмегімен атаулы семафорға ғана қол жеткізуге болады, ал атаусыз семафорлар осы обьекттің және ағымдары құратын процесспен ғана қолданылады.

Семафорды босату үшін ReleaseSemaphore() функциясы қолданылады:

```
include <windows.h>
BOOL WINAPI ReleaseSemaphore(
   HANDLE hSemaphore,
   LONG lReleaseCount,
   LPLONG lpPreviousCount);
```

Параметр hSemaphore, босатылуы тиіс семафордың дескрипторын береді. Параметр lReleaseCount, семафордың ішкі есептегішін арттыратын, мәнді береді. Берілген мән 0-ден көп болуы қажет. Параметр lpPreviousCount, семафор есептегішінің алдыңғы мәндерін қайтару үшін қолданылады.

Windows – те процесстер мен ағымдарды синхрондаудың бірі мьютекс болып табылады. Мьютекс өзімен, бір ағымға да жатпайтын болғанда, сигналды күйге ауысатын, синхрондау обьектісін ұсынады. Егер де мьютекс қандай да бір ағымға жататын болса, онда сигналды емес күйге ауысады. Осы тұрғыда, мьютекстер, олармен бөлінетін ресурсқа бірнеше ағымның қол жеткізуін болдырмауды ұйымдастыру кезінде, ыңғайлы. Мұндай ресурсқа мысал болып, бөлінетін жады сияқты, процессаралық өзара әрекеттесудің объектісі болып табылады. Осы объектіге ақпараттарды жазуға, әр уақыт сәтінде бір есептеуші ағым міндетті. Сондықтан, мұндай процессаралық өзара әрекеттесудің механизмінің қолжетімділігін болдырмауды ұйымдастырудың міндеті өте маңызды болып табылады. Жалпы мьютекстерді семафордың бір

нұсқасы ретінде қарастыруға болады.

Мьютексті құру үшін CreateMutexQ функциясы қолданылады:

```
#include <windows.h>
HANDLE WINAPI CreateMutex(
   LPSECURITY_ATTRIBUTES lpMutexAttributes,
   BOOL bInitialOwner,
   LPCTSTR lpName);
```

Параметр lpMutexAttributes топтарымен құрылатын мьютекс дескрипторы мұралануға, сондай-ақ оның қолжеткізу құқығын өзгертілуіне жауап береді. Егер осы параметр NULL мәніне ие болса, дескриптор мұраланбайды және әдеттегідей құқықтары қондырылады.

Егер bInitialOwner параметрі TRUE тең болса, онда мьютекс құрған ағым, пайдаланушымен жарияланады. Егер де осы параметр FALSE тең болса, онда мьютексті құратын есептеуші ағым, оны иелікке алмайды.

LpName параметрі, мьютекстің атауын беру үшін қолданылады. Егер осы параметр NULL тең болса анонимді объект құрылады. Егер Windows – те оқиға, семафор, мьютекс т.б. сияқты жүйелік обьектілерін сәтсіз құруда, функция қатемен аяқталады. Егер функция сәтті аяқталса және семафор құрылса CreateMutexQ функциясы құрылған объектінің дескрипторын қайтарады. Басқа жағдайда функция NULL мәнін қайтарады.

Жүйеде құрылған мьютекстердің максималды саны, жүйедегі бос ресурстардың болуымен (бос виртуалды жадының ерекшелігі) және процестегі дескриптор санының шектелуімен анықталады.

Алдымен құрылған мьютекстің дескрипторын алу қажеттілігін, жүйеде нақты көрсету үшін OpenMutex() функциясы қолданылады:

```
include <windows.h>
HANDLE WINAPI OpenMutex(
   DWORD dwDesiredAccess,
   BOOL bInheritHandle,
   LPCTSTR lpName);
```

DwDesiredAccess параметрі пайдаланушының мьютексті басқару үшін қандай қолжетімділік құқықтарын талап ететінін жүйеге хабарлайды. Осы параметр обьектіге толық қол жеткізу үшін MUTEX_ALL_ACCESS немесе MUTEX_ MODIFY_STATE мәндерін қабылдай алады.

BInheritHandle параметрі, мьютекстің құрылатын дескрипторы еншілес процеске мұрағатталатынын анықтайды. Егер бұл параметр TRUE мәніне ие болса, құрылатын дескриптор еншілес процесті мұралайды, басқа жағдайда мұраланбайтын дескриптор құрылады.

Параметр lpName қолжетімділікті алуы керек мьютекстің атауын береді. Осы функцияның көмегімен, тек атаулы семафорға қолжетімділік алуға болады, ал атаусыз семафорлар, тек берілген обьект және оның тобы құратын процеспен ғана пайдаланылады.

Мьютексті босату үшін ReleaseMutex() функциясы қолданылады:

```
include <windows.h>
BOOL WINAPI ReleaseMutex(HANDLE hMutex);
```

Параметр hSemaphore босатылуы қажет мьютекстің дескрипторын береді.

Есептеуші ағымдарды синхрондау үшін мьютекстер мен семафорларды пайдалану үлгісін қарастырамыз. Ол үшін алдында қарастырылған үлгіні жетілдіреміз және синхрондаудың екі механизмін қолданамыз: семафорлар және мьютекстер.

SemaphoreAndMutex.c

```
#include <windows.h>
#include <stdio.h>
#include <tchar.h>

DWORD WINAPI ThreadProc(LPVOID lpParam);
void ErrorReport(LPTSTR lpszFunction);

int main()
{
   DWORD dwThreadId;
   HANDLE hThread, hSem, hMutex;
   unsigned i;

   // сигналды емес бастамамен семафор құрамыз
     // мьютексті құруда процестерді синхрондау үшін
     // осы семафор қолданылады if((hSem =

     CreateSemaphore(NULL, 0, 1, "Threadl")) == NULL)
     {
        ErrorReport(TEXT("CreateSemaphore()")); return(1);
     }
```

```
hThread = CreateThread(
  NULL,       //  әдеттегідей құқық // әдепкә
  бойынша қамшы өлшемі
  ThreadProc, // ағым функциясы
  NULL,       //  функция үшін  аргумент жоқ
  0,          //  әдеттегідей жалаушалар
                &dwThreadId);
if(hThread == NULL)

{
  ErrorReport(TEXT("CreateThread()"));
  return(1);
}
// мьютексті еншілес етіп құру
// есептеуіш ағымымен WaitForSingleObject(hSem,
INFINITE);
// мьютекс құруда дескриптор аламыз if((hMutex =
OpenMutex(MUTEX_ALL_ACCESS, FALSE, "Thread2")) ==
NULL)
{
  ErrorReport(TEXT("OpenMutex()"));
  return(1);
}
// алғашқы ағымның негізгі  циклі for(i = 0; i < 10;
++i)
{
  // басқару жіберілімін күту
  // туындаған ағымнан
  WaitForSingleObject(hMutex, INFINITE);
  // ақпараттарды консолды
  шығару  printf("p%d ", i);
  // алғашқы ағымды 1 сек блоктау Sleep(1000);
  // туындаған ағымның басқарылуын жіберу
  ReleaseMutex(hMutex);
}
// туындаған ағымның аяқталуын күту
WaitForSingleObject(hThread, INFINITE);
//  дескриптор жабу CloseHandle(hThread);
// семафор және мьютекстің дескрипторын жабу
CloseHandle(hSem);
CloseHandle(hMutex);
```

```
   printf("\n");
   return(0);
}

DWORD WINAPI ThreadProc(LPVOID lpParam)
{
   HANDLE hSem, hMutex; unsigned i;
   // ертерек семафорын  дескриптор алу if((hSem =
   OpenSemaphore(SEMAPHORE_ALL_ACCESS, FALSE,
   "Thread1")) == NULL)
   {
      ErrorReport(TEXT("OpenSemaphore()")); return(1);
   }
   // бастапқы сигналды емес мьютексті құру
   //
   if((hMutex =CreateMutex(NULL, FALSE, "Thread2")) ==
      NULL)
   {
      ErrorReport(TEXT("CreateMutex()"));
      return(1);
   }
   // семафорды  сигналды күйге ауыстыру
   // ақпаратты шығаруды синхрондау үшін мьютексты
   құрылуы туралы алғашқы ағымына хабарлау
   // ReleaseSemaphore(hSem, 1, NULL);

   // еншілес ағымның негізгі циклі
   for(i = 0; i < 10; ++i)
   {
      // бастапқы ағымның басқаруды жіберуін күту
      //
      WaitForSingleObject(hMutex, INFINITE);
      // ақпараттарды  консолға шығару
    printf("c%d ", i);
    // ағымды 4 секундқа
    блоктауSleep(4000);
    // алғашқы ағымға басқаруды жібереміз
    ReleaseMutex(hMutex);
}
// семафор мен  мьютекстің дескрипторын жабамыз
CloseHandle(hSem);
```

```
   CloseHandle(hMutex); return(0);
}
void ErrorReport(LPTSTR lpszFunction)
{
   LPVOID lpMsgBuf;
   LPVOID lpDisplayBuf;
   DWORD dw = GetLastError();

   FormatMessage(
      FORMAT_MESSAGE_ALLOCATE_BUFFER |
      FORMAT_MESSAGE_FROM_SYSTEM,
      NULL,
      dw,
      MAKELANGID(LANG_NEUTRAL, SUBLANG_DEFAULT), (LPTSTR)
      &lpMsgBuf,
      0, NULL );

   lpDisplayBuf = (LPVOID)LocalAlloc(LMEM_ZEROINIT,
   (lstrlen((LPCTSTR)lpMsgBuf)+lstrlen((LPCTSTR)
   lpszFunction)+40)*sizeof(TCHAR));
   _stprintf((LPTSTR)lpDisplayBuf,
      TEXT("%s failed with error %d: %s"), lpszFunction,
   dw, lpMsgBuf);
   MessageBox(NULL, (LPCTSTR)lpDisplayBuf,
      TEXT("Error"), MB_OK);

   LocalFree(lpMsgBuf);
   LocalFree(lpDisplayBuf);
}
```

Берілген үлгіде семафор тек процестердің алғашқы синхрондау үшін қолданылады. Семафордың көмегімен, еншілес ағым алғашқы ағымға мьютекстің құрылуын хабарлайды, ол алғашқы және еншілес есептеуіш ағымдарды консолға қол жеткізуін болдырмауды ұйымдастырады. Ары қарай семафор қолданылмайды, тек мьютекс қолданылады.

Берілген бағдарламаның жұмысы нәтижесінде консолға келесі жол шығады:

```
c0 p0 cl pi c2 p2 c3 p3 c4 p4 c5 p5 c6 p6 c7 p7 c8 p8 c9
p9
```

Алдындағыдай, екі ағым үйлестірілген және олардың шығуы қатаң кезектеседі. Егер де жоғарыда келтірілген бағдарламаға ResetMutex() және WaitForSingleObjectQ функцияларының шақыртуларында ішкі циклдері және бастапқы ағым, ағым тобын түсіндіре кетсек, бағдарлама жұмысының нәтижесі мынадай болады:

```
c0 p0 p1 p2 p3 c1 p4 p5 p6 p7 p8 c2 p9 c3 c4 c5 c6 c7 c8
c9
```

Мұнда тағы жағдай қайталанады, ағымдардың синхрондауы жүргізілмесе бастапқы ағым өзінің жұмысын еншілес ағымнан бұрын аяқтайды және консолға ақпараттарды шығарғанда екі есептеуіш ағымдар синхрондалмайды.

БАҚЫЛАУ СҰРАҚТАРЫ

1. Үзілістер мен сигналдар механизмінің айырмашылықтары қандай?
2. Процеспен алынған сигналды өңдеу қалай жүргізіледі? Сигналды өңдеу үшін пайдаланушының мүмкіндіктері қандай?
3. Күтілетін нәтижеден өлшемі үлкен, хабарламалар кезектен қалай қабылданады?
4. Бинарлы семафор мен есептегіш-семафордың негізгі айырмашылықтары қандай?
5. Жалпы жадыны қолдану кезінде құралдардың синхрондалуын пайдалану не үшін қажет?
6. Атаусыз каналдар атаулы каналдардан несімен ерекшеленеді? Процесс каналдан ақпараттарды қалай жазады және қалай оқи алады?

ҚОСЫМША 1

«БІЛІМДІ БАҚЫЛАУ». КАТАЛОГТАР ҚҰРЫЛЫМЫ

«Білімді бақылау» жүйелердің әзірлемесінің міндеттерін қою 1.5. тарауда келтірілген.

«Білімді бақылау» каталогтары мен файлдары *check* каталогында орналасқан және келесі құрылым мен қол жеткізу құқықтарына ие:

Students каталогында студенттердің жұмыс аумақтары орналасқан.

Teacher каталогында бақылау жұмыстарының нұсқалар базасынан және жиналған жұмыстары тұратын, оқытушының жұмыс аумағы сақталады. Бақылау жұмыстарының нұсқалар базасы каталог жүйелерінен тұрады, олардың әрі бір тақырыпқа сәйкес келеді. Тақырып каталогының атауы — *theme<N>*, мұнда *N* — 1 –ден бастап тақырып жинағы және әрі қарай. Бақылау жұмысының әр нұсқасы атаулы файлды ұсынады *var<N>.txt*, мұнда *N* — 1-ден бастап әрі қарай нұсқа нөмірі. Файлдың бірінші жолы тақырып нөмірі мен нұсқаның нөмірінен тұрады.

Бақылау жұмысы нұсқалары файлының үлгісі:

```
1 тақырып " С тіліндегі ақпараттар құрылымы"

1 нұсқа
1 - сұрақ: Жадының қандай көлемі signed int айнымалы
типін алады ?
Жауабы: __

2- сұрақ: келесі құрылым жадының қандай көлемін алады?

    int       a;
    char[19]  b;
    float     c;
} example_struct;
packed struct {
```

Жауабы:_____

жиналған жұмыстардың каталогы студенттермен орындалған бақылау жұмыстарының нұсқалар файлынан тұрады. Файл атауының параметрі:<Аты>- theme<тақырып нөмірі>_гаг<нұсқа нөмірі>.£хк Мысалы, студентпен орындалған, екінші тақырыптың бірінші нұсқасы *vasya,* былай аталады: *vasya- theme2_var1. txt.*

Жүйемен үш түрлі пайдаланушы жұмыс жасайды:

- **өңдеуші** —*devel міркеме атауы, teacher* тобының мүшесі
- **оқытушы** — тіркеме атауы *teacher,* топ мүшесі *teacher;*
- **студенттер** — тіркеменің еркін атаулары, *teacher* топқа кірмейді.

Жеке топтарды белгілеу есебінен *teacher* бақылау жұмыстары бар каталогқа, студенттердің келісімсіз қол жеткізуін алдын алу мүмкін болады, ал қол жеткізу құқығын шектеу көшіріп жазу мүмкіндігін минимумға жеткізеді (басқа студенттің бақылау жұмысын қарау).

«Білімді бақылау» жүйелердің әзірлемесінің міндеттерін қою 1.5. тарауда келтірілген.

«Білімді бақылау» жүйесі шегінде орындалатын жұмыстарды қамтамасыз етудегі, тапсырмалар командалық интерпретатор BASH тілінде жазылған. Тапсырмаларды үш топқа жіктеу ұсынылған:

- check/ scripts/teacher каталогында орналасқан оқытушының жұмыстары үшін
- check/scripts/ students каталогында орналасқан студенттердің жұмыстары үшін;
- check/scripts каталогында орналасқан қызметтік тапсырмалар.

Оқытушылар мен студенттердің тапсырмасын дұрыс жіберуді қамтамасыз ету үшін BASEDIR айнымалы шеңбердің мәндерін тағайындау керек, ол «Бақылау» жүйесінің толық атауын береді. Студенттер қолданатын пакеттік файлдар үшін студенттің жұмыс облысын сақтайтын және каталог атауын көрсететін, NAME айнымалысының мәндерін орнату керек. Ол үшін *scripts/env.sh*

тапсырмасы қолданылады. «Бақылау» жүйесінің құрамына кіретін, кейбір пакеттік файлды мәтіндерінің үлгілері келтірілген.

Тапсырмалардың бастапқы мәтінінің мысалдары :

scripts/env.sh — айнымалы шеңберінің қондырма пакеті

```
#!/bin/bash

# айнымалыны орнату үшін арналған тапсырма
# жүйемен дұрыс жасау үшін пайдаланушының шеңбері
# if [ " ${EDITOR:-DUMMY} " == "DUMMY" ] ; then
export EDITOR=mcedit echo "EDITOR=$EDITOR" fi
export BASEDIR=/check/ echo "BASEDIR=$BASEDIR" export
NAME='whoami' echo "NAME=$NAME"
export PATH=$PATH:$BASEDIR/scripts/teacher: \
$BASEDIR/scripts/students
echo " PATH=$PATH"
```

scripts/rights.sh — жүйенің каталогына қол жеткізу құқығын орнату пакеті

```
#!/bin/bash

# жүйенің каталогы қол жеткізу құқығын орнатуға

арналған тапсырмалар

# негізгі каталогтарға қол жеткізу құқығын орнату

chown devel.teacher $BASEDIR
chmod 755 $BASEDIR
chown -R devel.teacher $BASEDIR/scripts chmod -R 755
$BASEDIR/scripts chown devel.teacher
$BASEDIR/students chmod 755 $BASEDIR/students

TEACHERDIR=$BASEDIR/teacher chown teacher.teacher
$TEACHERDIR chmod 7 00 $TEACHERDIR

# каталогтардың каталогшаларына қол жеткізу құқығын
орнату
# оқытушылар. Иесіне және топқа қол жеткізу
# (яғни оқытушы мен өңдеушіге арналған) for i in 'ls
$TEACHERDIR' ; do
    chown teacher.teacher
$TEACHERDIR/$i chmod 770
$TEACHERDIR/$i done

# Студенттердің каталогына қол жеткізу құқығын орнату
# Әр студент – өз каталогының иесі
```

```
# Сонымен қатар teacher тобының мүшесі жаза алады (оқу
емес)
#  студент каталогына және каталогты оқу ready әр
студент
for i in 'ls $BASEDIR/students' ; do
STUDENTDIR=$BASEDIR/students/$i
READYDIR=$STUDENTDIR/ready if [ -d $STUDENTDIR ] ;
then chown $i.teacher $STUDENTDIR chmod 730
$STUDENTDIR fi
    if [ -d $READYDIR ]; then
    chown $i.teacher $STUDENTDIR
    chmod 770 $STUDENTDIR fi
done
```

scripts/teacher/makedirs.sh — жүйенің каталогтарын құру пакеті

```
#!/bin/bash

# "Бақылау жұмыстары" бастапқы құруларға арналған
тапсырмалар
# mkdir -p $BASEDIR жүйесінің барлық каталогтарына
mkdir $BASEDIR/scripts mkdir
$BASEDIR/scripts/students mkdir
$BASEDIR/scripts/teacher mkdir $BASEDIR/students
mkdir $BASEDIR/students/fedya mkdir
$BASEDIR/students/fedya/ready mkdir
$BASEDIR/students/ivan mkdir
$BASEDIR/students/ivan/ready mkdir
$BASEDIR/students/kolya mkdir
$BASEDIR/students/kolya/ready mkdir
$BASEDIR/students/petr mkdir
$BASEDIR/students/petr/ready mkdir $BASEDIR/teacher
mkdir $BASEDIR/teacher/theme1 mkdir
$BASEDIR/teacher/theme2 mkdir $BASEDIR/teacher/theme3
mkdir $BASEDIR/teacher/theme4 mkdir
$BASEDIR/teacher/works
```

scripts/teacher/give.sh — студентке тапсырма беру пакеті

```
#!/bin/bash

# белгілі бір студентке берілген тапсырма бойынша

нұсқаларды таратуға арналған тапсырма
#
#   шақырту параметрлері:
#   $1 – тақырып нөмірі
#   $2 – нұсқа нөмірі
#   $3 – студенттің тіркеме атауы
```

```
# Шеңбердің жұмыс айнымалылары:
# BASEDIR – жүйенің негізгі каталогы

# Алғашқы тексерістер ##################################

if [ " ${BASEDIR:-DUMMY} " == "DUMMY" ] ; then echo
"айнымалы \$BASEDIR берілмеген" exit 100

fi

if [ ! -d $BASEDIR -o ! -r $BASEDIR -o ! -x $BASEDIR ]
then
   echo "$BASEDIR каталог емес және қолжетімсіз" exit
101

   fi

if [ $# -ne 3 ]; then
   echo "шақырту параметрі:"
   echo " 'basename $0' <N тақырып> <N нұсқа> \
         <студент аты>"
   exit 1

   fi

THEMEDIR=$BASEDIR/teacher/theme$1 if [ ! -d $THEMEDIR ];
then
   echo " $1"

   exit 2

   fi

   тақырыбымен каталогты ашу мүмкін емес

if [ ! -r $THEMEDIR -o ! -x $THEMEDIR ] ; then
   echo " $1 тақырыпты каталог оқуға қолжетімсіз \
немесе орындауға"

   exit 3

   fi

VARIANTNAME=$THEMEDIR/var$2.txt
if [ ! -f $VARIANTNAME ] ; then
   echo " $2 нұсқасының файлы $1 жоқ"

   exit 3

   fi

if [ ! -r $VARIANTNAME ] ; then
   echo " $2 нұсқа файлы $1 тақырыбында қолжетімсіз \
оқуға"
```

```
   exit 4
   fi

STUDENTDIR=$BASEDIR/students/$3
   if [ ! -d $STUDENTDIR ]; then
   echo " студенттің жұмыс каталогы жоқ $3"
exit 5

   fi

if [ ! -w $STUDENTDIR ]; then
   echo " Студенттің $3 жұмыс каталогы
қолжетімсіз \ жазу бойынша"
   exit 5
fi

# негізгі бөлім ########################################

cp $VARIANTNAME $STUDENTDIR/theme$1_var$2.txt RETCODE=$?

if [ $RETCODE -ne 0 ] ; then
   echo "файлды көшіру мүмкін емес $1\ нұсқа $2
        студенттің тақырыбы $3"
   RETCODE='expr $RETCODE + 100'

   exit $RETCODE

fi

chmod 666 $STUDENTDIR/theme$1_var$2.txt RETCODE=$?

if [ $RETCODE -ne 0 ] ; then
   echo " студенттің $1 тақырыбы $2 нұсқасының файлы
        үшін қол жеткізу құқығын орнату мүмкін емес$3"
   RETCODE='expr $RETCODE +
   50' exit $RETCODE

fi
```

scripts/teacher/give_all.sh — барлық студенттерге бақылау тапсырмаларын тарату пакеті

```
#!/bin/bash

# Тапсырмалар нұсқаларын таратуға арналған тапсырма
# Барлық студенттерге тапсырылған тақырып бойынша
#
# Шақырту параметрлері:
# $1 – тақырып нөмірі
# Шеңбердің жұмыс айнымалылары:
# BASEDIR – жүйенің негізгі каталогы
```

```
# Бастапқы тексерістер
####################################

if [ $# -ne 1 ] ; then
   echo "шақырту параметрі" echo
"'basename $0' <тақырып нөмірі>"
exit 1

   fi

if [ " ${BASEDIR:-DUMMY} " == "DUMMY" ] ; then echo
"Айнымалы \$BASEDIR тапсырылған"

exit 100

fi

if [ ! -d $BASEDIR -o ! -r $BASEDIR -o ! -x $BASEDIR ]
then
   echo "$BASEDIR каталог емес немесе қол жетімсіз "
exit 101

   fi

# Негізгі бөлім ########################################

NUM_VARIANTS='$BASEDIR/scripts/teacher/look.sh $1'

if [ " ${NUM_VARIANTS:-DUMMY} " == "DUMMY" ] ; then echo
" нұсқалардың жалпы санын алу мүмкін емес "

exit 201

fi

i=1 # нұсқа нөмірінің есептеуіші

for j in 'ls $BASEDIR/students' ; do
   # Студент атауының шығуы және нұсқасының
   нөмірі echo "Студентке $j $1тақырыбы
   бойынша $i нұсқасы берілді "
   # Тапсырманы іске асыру give.sh тапсырм
   аларды тарату үшін
   $BASEDIR/scripts/teacher/give.sh $1 $i $j
   i='expr $i + 1'

   # Егер есептегіш  максималды нөмірге жетсе,
   if [ $i -gt $NUM_VARIANTS ] ; then
     i=1 # тастаймыз fi
done
```

scripts/teacher/gather.sh — орындалған бақылаудың жинағы

```
#!/bin/bash
```

```
# Студенттің жұмыс каталогындағы ready каталогшасынан

орындалған бақылауларды жинауға арналған тапсырма
#
# Шақырту параметрі : жоқ
# Жұмыс айнымалы шеңбері:
# BASEDIR – жүйенің негізгі каталогы # Начальные
проверки ################################### if [ "
${BASEDIR:-DUMMY} " == "DUMMY" ] ; then echo "айнымалы
\$BASEDIR берілмеген" exit 100 fi

if [ ! -d $BASEDIR -o ! -r $BASEDIR -o ! -x $BASEDIR ]
then
   echo "$BASEDIR каталог емес және қол жетімсіз "
exit 101

   fi

WORKSDIR=$BASEDIR/teacher/works
if [ ! -d $WORKSDIR -o ! -r $WORKSDIR -o ! \
       -x $WORKSDIR ]; then
   echo "$WORKSDIR каталог емес немесе қолжетімсіз "
exit 101

   fi

# негізгі бөлім ########################################

for i in 'ls $BASEDIR/students' ; do
   READYDIR=$BASEDIR/students/$i/ready if [ ! -d
   $READYDIR -o \
        ! -x $READYDIR -o \
        ! -r $READYDIR ] ; then
      echo " ready каталогы студентте  $i жоқ немесе
           қолжетімсіз \ "
   else
      for j in 'ls $READYDIR' ; do
         mv $READYDIR/$j $WORKSDIR/$i-$j

         RETCODE=$?
         if [ $RETCODE -ne 0 ] ; then
            echo "студентте $j жұмысымен файл
                 қолжетімсіз $i "
         else
            echo "студенттен $i \ $j"
         жұмысымен файл алынды fi
      done
   fi
done
```

UNIX КОМАНДАЛАРЫ БОЙЫНША ҚЫСҚАША АНЫҚТАМА

UNIX операциялық жүйелерінің негізгі командалары бойынша анықтамалық ақпарат П.1.кестеде келтірілген. Көптеген ақпаратты әр команда үшін *man* анықтамалық жүйесінің парағын шақырып, алуға болады. Егер команда *bash* ішкі командалық сияқты белгіленген болса, (оның атауымен (BASH) белгісі жасалған) ол туралы ақпаратты *bash* анықтамалық жүйесі парағын шақырып, алуға болады.

Шартты белгілер: шаршы жақшада [ ] міндетті емес құрастырылымдар көрсетілген, мысалы, міндетті емес параметрлер,

П1 Кесте. UNIXжүйесінің негізгі командалар параметрі

Команда	Мазмұны
(BASH)	Командалық интерпретатордың сондай көшірмесінен BASH тапсырмасын іске асыру. **Шақырту параметрі:** . <тапсырма атауы> **Параметрлер:** <тапсырма атауы> — іске асырылатын тапсырманың атауы. **Пайдалану үлгісі:** . /usr/bin/purge script.sh
(BASH)	Бос команда, қайтару коды әрқашан 0 тең. **Шақырту параметрі:** **Параметрлер:** жоқ. **Қолдану үлгісі:** while : ; do echo "Infinite loop" done
break (BASH)	Циклден шығу for немесе while. **Шақырту параметрі:** break [n] **Параметрлер**: *n* — шығыс болатын енгізілген циклдердің саны

бұрыштық жақшада< > — міндетті параметрлер.

Команда	Мазмұны
	Қолдану үлгісі: for i in 'ls' ; do cat $i if ["$i" == "end"] ; then break fi done
cat	Стандартты шығару ағымының құрамдас файлын шығару. **Шақырту параметрі:** cat [параметрлер] <файл атауы> **Параметрлер:** -s — бір болып, файлда қатар жүру үшін бірнеше бос жолды алмастыру; -E — әр жолдың аяқталған соң $ символын қарау. **Қолдану үлгісі** cat -s /home/sergey/file.txt
cd, chdir (BASH)	Көрсетілген каталогқа өту. **Шақырту параметрі** cd [каталог атауы] **Параметрлер:** [каталог атауы] — алмасу болатын каталогтың қатысты немесе толық атауы. Егер параметр берілмесе немес атауы берілсе"~", пайдаланушының үй каталогына алмасады. Егер атауы ретінде "~<пайдаланушының тіркеме атауы>" берілсе, осы пайдаланушының үй каталогына алмастыру орын алады (жеткілікті құқықтары болғанда). **Қолдану үлгісі:** cd /usr/local/bin cd ~alex
chgrp	Файлға берілген иелер - топтарын өзгерту. **Шақырту параметрі:** chgrp [параметрлер] <топ> <тізім> **Параметрлер:** *-R* — тізімде көрсетілген, каталогтардың каталогшаларына пайдаланушылардың рекурсивті өзгеруін енгізу; -v — әр өңделетін файл мен каталогтың атауларын экранға шақыру;

Команда	мазмұны
	-c — иелер-тобы өзгеретін, әр файл немесе каталог атауларын экранға шығару; <топ> — атауы немесе GID иелер – топтары, қондырылуы керек <тізім> — жаңа иелер-топтарын орнату үшін файлдар мен каталогтар аттарының тізімі. **Қолдану үлгісі:** chgrp -R -v users /home/vasya
chown	Берілген файлдарға арналған пайдаланушы-иесін өзгерту. **Шақырту параметрі:** chown [параметрлер] <пайдаланушы> <тізім> **Параметрлер:** -R — тізімде көрсетілген, каталогтардың барлық каталогшаларына иелерін рекурсивті өзгерту; -v —әр өңделетін файл мен каталогтың атауын экранға шығару; -c — иелер-тобы өзгеретін, әр файл мен каталогтардың атауын экранға шығару; <пайдаланушы> — пайдаланушы-иесінің UID немесе атауы орнатылған; <тізім> — жаңа пайдаланушы-иесі орнату файлдар мен каталогтардың аттар тізімі; **Қолдану үлгісі:** chown -R -v vasya /home/vasya
cp	Файлдар мен каталогтарды көшіру. **Шақырту параметрі:** cp [параметрлер] <файлдар> <каталог> —Тізімнен каталогқа файлды көшіру; cp [параметрлер] <файл1> <файл2> —Файлды файлға көшіру; файл1 атымен файл2. **Параметрлер:** -r — каталогтарды рекурсивті көшіру; -v — көшірудің алдында әр файлдың атын шығару. **Қолдану үлгісі:** cp -r file1.txt file2.txt /usr/doc/
cut	Файлдың форматталған жолынан жеке жолақтарды алып шығару. **Шақырту параметрі:** cut -f <өріс нөмірі> ^<бөлу>

Команда	Мазмұны
	Параметрлер: -f — алынатын жолақтың нөмірін береді; -d — форматталған жолақтың бөлгіштігін береді. **Қолдану үлгісі:** *file.txt* құрамымен файлы үшін Иванов:Иван:Иванович:1978 команда cat file.txt | cut -f2 -d: шығарады Иван
diff	Екі файлдың айырмашылықтарын іздеу және шығарудың стандартты ағымы. **Шақыртудың параметрі:** diff [параметрлер] <файл1> <файл2> **Параметрлер:** -b — бос орындарда және табуляция символдарында айырмашылықтарды елемеу; -t — шығаруда бос орындар мен табуляциядағы символдарды алмастыру; -u — шығарудың бірыңғайланған параметрін қолдану; -n —RCS-diff параметрінде шығару. **Қолдану үлгісі:** diff -nur file1.txt file2.txt
echo (BASH)	Шығарудың стандартты ағымына мәтіннің жолдарын шығару; **Шақырту форматы:** echo [параметрлер] <мәтін жолы> **Параметрлер:** *-n* — жолды шығарудан кейін, жолдың символ ауыстырылуын шығармау **Қолдану үлгісі:** echo "Hello world"
exec (BASH)	Қазіргі командалық интерпретатор процесіне оның процесін алмастырумен бағдарламаны орындау **Шақырту параметрі:** exec <бағдарламаның атауы> **Параметрлер:** <бағдарламаның атауы> —орындалатын файлдың қысқаша, қатынасты, толық атауы. **Қолдану үлгісі:** exec ls

Команда	Мазмұны
	Бастапқы командалық интерпретатормен орындалған, қазіргі каталогтан файлдар тізімін шығарады, одан кейін сеанс аяқталады, (өзімен бастапқы интерпретаторды ауыстырғандықтан)
exit (BASH)	Қайтары коды берілген қазіргі тапсырманың орындалуын аяқтау. **Шақырту параметрі:** exit <n> **Параметрлер:** <n> — қайтару коды. Теріс емес сан **Қолдану үлгісі:** exit 1
export (BASH)	Осы тапсырманың ішкі ортасына, жарияланған тапсырмалардың айнымалыларының орынын ауыстыру. **Шақырту параметрі:** export <айнымалы атауы> export <айнымалының атауы>=<мәні> **Параметрлер:** <айнымалының атауы> — орта шығарылатын айнымалының аты; <мәні> — ортаға шығарылмас бұрын айнымалымен тағайындалған мән. **Қолдану үлгісі:** export IP Address=192.168.0.1
grep	Экранда табылған жолдарымен файлдағы мәндер немесе жолдарды іздеу. **Шақырту параметрі:** grep [параметрлер] <жол> <файл тізімі> **Параметрлер:** -c —жолда құрылған жолдардың санын береді; -i — іздеу кезінде символдарды тіркеуді елемейді; -n — әр жолдың алдында оның файлдағы нөмірін береді; <жол > — іздеуге арналған жиі мәндер немесе символдардың жолы; <файлдар тізімі> — іздеу орын алатын, файлдар атауының тізімі . Команда 0 қайтарады, егер жол табылса, және 1, егер табылмаса.

Команда	Мазмұны
	Қолдану үлгісі: grep "Операциялық жүйелер" book.txt if [$? -ne 0] ; then echo "жол табылмаса" fi
gzip	LZW алгоритмін қолданып файлды сығымдау. Сығымдалған файл, кіріс файлы сияқты сол атауға ие болады, бірақ *gz* кеңеюіне ие. Көп файлды бір архивке сығымдау үшін алдымен файлдарды *tar* командасының көмегімен желімдеу қажет. **Шақырту параметрі:** gzip [параметрлер] <файлдың атауы> **Параметрлер:** <файлдың атауы> — сығымдалатын файлдың атауы; -*c* — кіріс файлды өзгертпей, сығымдалған ақпараттарды, стандартты шығару файлына салу; -t — сығымдалған файлдың бүтіндігін тексеру ; -d — сығымдалған файлды ашу; -v — сығымдалатын файлдың атауын және оның сығымдалған үлесін шығару. **Қолдану үлгісі:** gzip -d linux-kernel-2.4.34.tar.gz
head	Файлдың алғашқы жолдарын шығару. **Шақырту параметрі:** head [параметрлер] <имя файла> **Параметрлер:** <имя файла> — алғашқы жолдары шығарылатын файлдың атауы; -n <число строк> —файлдың басынан берілген жолдар санын шығару; -c <число байтов> — файлдың басынан берілген байттар санын шығару. Теріс санды *n* көрсеткенде, жолдар немесе байттар барлық мәтінге шығады, соңғы *n* жолдар немесе байттардан басқа. **Қолдану үлгісі:** head -n 15 file.txt
kill	PID берілген процеске сигналды тасымалдау. **Шақырту параметрі:** kill [параметрлер] <PID>

Команда	Мазмұны
	Параметрлер: *-l* — қолжетімді сигналдардың тізімін шығарады; -<нөмір> немесе -<атауы> — тасымалданатын сигнал; <PID> — PID сигналдар тасымалданатын процесстер. **Қолдану үлгісі:** kill -SIGHUP 1035
killall	Белгілі атаулы бағдарламаларды іске асырудың нәтижесінде құрылған, барлық процестерге сигнал тарату. **Шақырту параметрі:** killall -<сигнал> <бағдарлама аты> **Параметрлер:** -<сигнал> — нөмірмен немесе атаумен берілген тасымалдаушы сигнал; <бағдарлама атауы> — процесті тудырушы бағдарламаның атауы. *ls* командасымен шығарылған, процестер кестесінде оқылуы мүмкін. **Қолдану үлгісі:** killall -SIGSTOP hasher
less	Айналдыру және іздеу мүмкіндігі бар парақшамен мәтіндік файлды шығару. **Шақырту параметрі:** less < файл атауы> **Параметрлер:** <файл атауы> — шығаратын файлдың атауы. **Қолдану үлгісі:** less file.txt
ln	Файлда сілтемелер құру. **Шақырту параметрі:** ln [параметрлер] <кіріс файл> <сілтеме файлдар > **Параметрлер:** Параметрсіз іске асырылған команда <сілтеме файлдар> атымен катқыл сілтемені құрады <кіріс файл> аты көрсететін ақпараттар жинағы; *-s* — қатқыл сілтеменің орнына символдық сілтеме. **Қолдану үлгісі:** ln -s file.txt linkhle.txt

Команда	Мазмұны
ls	Каталог файлдарының тізімін шығару. **Шақырту параметрі:** ls [параметрлер] [файлдар мен каталогтар тізімі] **Параметрлер:** -a — "." басталатын, атаулы файлдарды шығару ; -l — файлдардың атрибуттары туралы кеңейтілген ақпаратты шығару; [файлдар мен каталогтардың тізімі] — *ls* командасымен шығарылатын файлдар тізімі, оның құрамы *ls* командасымен шығарылады. **Қолдану үлгісі:** ls -l /home/nick
mkdir	Каталогты құру. **Шақырту параметрі:** mkdir [параметрлер] < каталог атауы> **Параметрлер:** -p — егер <каталог атауы> бірнеше иерархияларды қосатын ретінде аты берілсе, (мысалы, */usr/local/share/doc/programs),* онда бұл кілтті көрсету кезінде, барлық жетіспейтін каталогтарды құрылатын болады, яғни */usr/local/share/doc* каталогы болмайды, *онда басында ол құрылады,* одан кейін оның каталогшасы *programs.* **Қолдану үлгісі:** mkdir -p /usr/local/share/doc/programs
more	Мәтіндік файлды парақтау мүмкіндігімен экранға парақтап шығару. **Шақырту параметрі:** more <файл аты> **Параметрлер:** <файл аты> — шығарылатын файл атауы. **Қолдану үлгісі:** more file.txt
mv	Файлдардың орын ауыстыруы (аттарын ауыстыру). **Шақырту параметрі:** mv [параметрлер] <кіріс файл> <файл арналуы> mv [параметрлер] <файл> <каталог арналуы> **Параметрлер:** -f — операцияларды растауға ешқандай сұраныс бермейді;

Команда	Мазмұны
	-i — файлдың арналуы болса растауға ешқандай сұраныс бермейді. Команда шақыруының бірінші формасында файл атын ауыстырады, атауы <исходный файл> параметрімен берілген, <файл арналуы> параметрімен берілген атты тағайындайды. Шақыртудың екінші формасы файлды каталогқа салады, <каталог арналуы> параметрімен беріледі. **Қолдану үлгісі:** mv file1.txt file2.txt mv file1.txt dir2/
nice	Тапсырмаларды жобалаушыға арналған бастапқы өзгерісімен бағдарламаны іске асыру. Басымдылық процеске, процестік уақыттың белгілеу жиілігін анықтайды. **Шақырту параметрі:** nice [параметрлер] <бағдарлама> [бағдарламаның аргументтері] **Параметрлер:** -n <жылжыту> — процестің басымдылық базасын, жылжыту бірлігіне өзгерту. Жылжыту 20-дан (жоғары басымдылық) 19-ға (төменгі басымдылық) дейін аралықты алады; <бағдарлама> — іске асырылатын бағдарламаның орындалатын файлының атауы; [бағдарламаның аргументтері] — іске қосқан кезде, бағдарламаға берілетін параметрлер, **Қолдану үлгісі:** nice -n 15 alpha gamma 1057
ps	Іске қосылған процестер туралы ақпаратты шығару. **Шақырту параметрі:** ps [параметрлер] **Параметрлер:** *a* — барлық пайдаланушылардың барлық процестері туралы ақпаратты шығару; *x* — терминалға байланбаған процестерді шығару (жүйелік процестер және демондар); *l* — шығарудың ұзын параметрі (максимум ақпарат); *u* — пайдаланушыға бағытталған шығару параметрі (ең қажетті ақпараттар); *s* — өңделетін сигналдар туралы ақпарат шығару. **Қолдану үлгісі:** ps aux

Команда	Мазмұны
pwd	Қазіргі каталогтың атауын шығару. **Шақырту параметрі:** pwd **Параметрлер:** жоқ. **Қолдану үлгісі:** pwd (мысалы, /home/sergey)
read (BASH)	Параметрде берілген айнымалылары сөздің мәнін тағайындау мен енгізудің стандартты ағымына жолдарды ауыстыру немесе табуляция символдарымен, бос орындарға бөлінген сөздерді жолы бойынша оқу. **Шақырту параметрі:** read <айнымалылардың тізімі> **Параметрлер:** <айнымалылардың тізімі> — үтір арқылы берілген, айнымалылардың тізімі **Қолдану үлгісі:** echo Иванов Иван Иванович | read namel name2 name3 жолдарды иеленуді шақырады *Иванов, Иван* и *Иванович* сәйкесінше айнымалылармен *namel, name2*и *name3*.
rm	Файлдар мен каталогтарды өшіру. **Шақырту параметрі:** rm [параметрлер] < файлдар мен каталогтардың тізімі> **Параметрлер:** *-f* —операцияның расталуын сұрамау; *-i* — әр файл мен каталогты өшіруге рұқсат сұрау; -r — берілгеннен бастап, барлық каталогтың діңгегін рекурсивті өшіру; < файлдар немесе каталогтар тізімі> —өшірілуге жататын файлдар мен каталогтардың атаулары. **Қолдану үлгісі:** rm -rf /home/nick/junk files/
rmdir	Бос каталогтарды өшіру (файлдар мен каталогшаларды құрамайды). **Шақырту параметрі:** rmdir [параметрлер] < каталог атауы>

Команда	Мазмұны

rmdir	**Параметрлер:** -p — егер каталог атауы өзіне бірнеше иерархия кезеңдерін (мысалы, */cat1/cat2/cat3)* қосатын болса, онда бұл кілтті көрсеткенде команда каталогтарды төменгі деңгейден бастап өшіреді, (яғни қаралатын жағдайда ұқсас болады *rmdir/cat/cat2/cat3; rmdir /cat1/cat2* ; *rmdir /catl);* < каталог атауы> — өшірілетін бос каталогтардың атауы. **Қолдану үлгісі:** rmdir -p /cat1/cat2/cat3/
set (BASH)	Ортада анықталған айнымалылар тізімін шығару немесе позициялық параметр мәнін тағайындау. **Шақырту параметрі:** set [мәндердің тізімі] **Параметрлер:** Параметрлері жоқ кезде олардың айнымалылары мен олардың мәндерінің толық тізімін шығарады; мәндер тізімін беру кезінде,бос орындармен бөлінген, ізінше құрылған айнымалыларды тағайындайды, $1 ... ${n}, мұнда *n* — мәндердің саны. **Қолдану үлгісі:** set 2 3 4 5 echo $3 (4 шығарылады)
shift (BASH)	Берілген орныққан жерлер санына, тірескен параметрлерді оқу үшін терезені жылжыту. **Шақырту параметрі:** shift [n] **Параметрлер:** [n] — жылжыту орын алатын, орныққан жерлер саны. Егер саны көрсетілмесе, онда жылжыту 1 орынға жүзеге асады. **Қолдану үлгісі:** тапсырмалар *a b c d e f* параметрлерімен іске асырылады. echo $1 (а шығарады) shift 4 echo $1 (е шығарады)
sleep	Секунд санына берілген тапсырма орындауды тоқтату. **Шақырту параметрі:** sleep <n>

Команда	Мазмұны
sleep	**Параметрлер:** <n> — тапсырмалардың орындалуы тоқтатылатын, секунд саны **Қолдану үлгісі:** sleep 10
sort	Енгізу стандартты ағымынан берілетін, жолдардың алфавиттік жинақтау. **Шақырту параметрі:** sort [параметрлер] **Параметрлер:** -r — алфавит ретіне қарсы жинақтау; -b — жолдың басында бос орындарды елемеу. **Қолдану үлгісі:** cat file.txt | sort > sorted.txt
tail	Файлдың соңғы жолдарын шығару. **Шақырту параметрі:** tail [параметрлер] <файл атауы> **Параметрлер:** < файл атауы> — соңғы жолдары шығарылатын файлдың атауы; -n <жолдар саны> — файлдың соңынан долдардың берілген санын шығару -c <байттар саны> — файлдың соңынан байттардың берілген санын шығару; -f — файлдың соңына жетуі бойынша бағдарламаның орындалуын аяқтамау, файлға ақпараттардың салынуын күту және түскеніне байланысты шығару. Жаңа жазбаларды файлдық хаттамаларға шығару ыңғайлы болуы мүмкін. Егер *-n* немесе *-c* параметрлерді көрсетпесе, онда команда, олардың шақыртуы кезінде файлға жазылған, ақпараттарды шығармай, ақпараттардың файлға қосылуын күтеді; -F — ұқсас -f, бірақ *tail* командасының жұмысы уақытында, файлдың атын ауыстыру жағдайын бақыланады.. Жолдардың немесе байттардың теріс санын көрсеткен кезде, *n* жолы немесе байттарының соңғысынан басқа барлық мәтін шығарылады.**Қолдану үлгісі:** tail -F logfile.txt
tar	Архивтеуге дайындау үшін каталогтар мен файлдарды бір файлға желімдеу.

Команда	Мазмұны
	Шақырту параметрі: tar <параметр>< параметрді жетілдіруші> <архивтік файлдың аты> <архивтеуге арналған файлдардың тізімі > **Параметрлер:** -c — жаңа архив құрылады; жазба соңғы файлдан емес, архивтің басынан басталады; -u — егер онда әлі болмаса, немесе берілген архивтің соңғы жазбасында өзгерген болса көрсетілген файлдар архивке салынады; -x — көрсетілген файлдар архивтен алынады; < архивті файлдың атауы> — архивтеуге жататын файлдарды желімдейтін, файл атауы; < архивтеуге арналған файлдар тізімі> — архивтеуге жататын файлдар мен каталогтардың атауының тізімі. Каталогтар рекурсивті өңделеді. **жетілдірушілер:** *z* —файлдарды бірден *gzip* архиватормен орамдау немесе шығару ; *v* — әр өңделетін файлдың атын шығаруды шығарады; *f* — егер осы жетілдіруші көрсетілсе және архивті файлдың орнына «-» (минус) символы көрсетілсе, онда архив оқылады немесе стандартты енгізу немесе шығару ағымына беріледі. **Қолдану үлгісі:** cd fromdir; tar -cf - . | (cd todir; tar -xf -)
tee	Бір кірісті конвейер (стандартты енгізу) және екі шығыс (стандартты шығару және көрсетілген файл). **Шақырту параметрі:** tee [параметрлер] <шығушы файл> **Параметрлер:** -*a* — шығушы файлдың соңына мәтінді қосу; <шығушы файл> — жазуға болатын файл. **Қолдану үлгісі:** cat infile.txt | tee -a outfile.txt
test	Шартты белгілердің мәнін есептеу. Толығырақ 5.8.6 және 7.2. тарауын қараңыз. **Шақырту параметрі:** test <белгілер>

Команда	Мазмұны
	Параметрлер: <белгілер> — тексерілетін шартты белгілер. **Қолдану үлгісі:** test -f file.txt
touch	Қазіргіге файлды көрсететін жетілдірудің мерзімін өзгерту немесе егер көрсетілген файл болмаса жаңа файлды құру. **Шақырту параметрі:** touch <файлдың атауы> **Параметрлер:** <файлдың атауы> —құрылатын файлдың атауы немесе өзгеретін мерзімді файл. **Қолдану үлгісі:** touch lock.txt
trap (BASH)	Берілген нөмірлі сигналды беруді қабылдаған кезде орындалатын командаларды анықтау. **Шақырту параметрі:** trap <команда> < сигналдар тізімі> **Параметрлер:** <команда> —орындалатын команда; <сигналдар тізімі> — сигнал нөмірінің тізімі **Қолдану үлгісі:** trap "exit 1" 3 4 7
unset (BASH)	Айнымалылардың деинициализациясы. Осы команданың орындалғанынан кейін, айнымалы енді нақты мәнге ие болмайды. **Шақырту параметрі:** unset <айнымалының аты> **Параметрлер:** <айнымалының аты> — деинициализацияланатын айнымалының мәні. **Қолдану үлгісі:** var="123" ; echo var ; unset var echo var (қате пайда болады)
wait (BASH)	Берілген PID процестің аяқталуын күту және оның қайтарым кодын қайтару. **Шақырту параметрі:** wait <PID>

Команда	Мазмұны
	Параметрлер: <PID> — команда аяқталуын күтетін PID процесстер. **Қолдану үлгісі:** wait 1078
wc	Файлдағы жолдар, сөздер немесе байттардың санын шығару; алдымен саны шығады, одан кейін бос орыннан соң—файлдың атауы. **Шақырту параметрі:** wc [параметрлер] <файл атауы> **Параметрлер:** -c — файлдағы байттар саны шығады; -l — файлдағы жолдар саны шығады; -w — файлдағы сөздер саны шығады; **Қолдану үлгісі:** wc -w file.txt | cut -f1 ^\<пробел>
which	Егер бұл каталог $PATH айнымалысында болса, атауы берілген бағдарламаға толық жол шығады. **Шақырту параметрі:** which <бағдарламаның аты> **Параметрлер:** <бағдарламаның аты> — каталог іздеу жүзеге асатын, орындалатын файлдың аты. **Қолдану үлгісі:** which ls (/bin)

ӘДЕБИЕТТЕР ТІЗІМІ

1. *Гордеев А.В.* Жүйелік бағдарламалық қамтамасыз ету / А. В. Гордеев, А. Ю. Молчанов. — СПб. : Питер, 2003. — 736 б.

2. Графикалық стандарт X Window. — М. : басп. ИМВС РАН, 2000. — 316 б.

3. *Дунаев С.* UNIX-сервер : В 2 т. / С. Дунаев. — М. : Диалог-МИФИ, 1998. — 304 б.

4. *Краковяк С.* ЭВМ ОЖ функциялау және ұйымдастыру негіздері / С. Краковяк. — М. : Мир, 1988. — 480 б.

5. *Орлов В.Н.* Мобильді операциялық жүйе МОС ЕС / В. Н. Орлов, В. Ю. Блажнов, О. А. Барвин. — М. : Қаржы және статистика, 1990. — 208 б.

6. *Померанц О.* Ядро Linux. Модульдерді бағдарламалау / О. Померанц. — М. : КУДИЦ-Образ, 2000. — 112 б.

7. *Робачевский А.М.* UNIX операциялық жүйесі / А. М. Робачевский. — СПб. : BHV-Петербург, 1999. — 528 б.

8. Жүйе әкімшісінің басшылығы / Э. Немет, Г. Снайдер, Е. Сибасе, Т. Хейн. — СПб. : Питер ; Киев : BHV, 2002. — 928 б.

9. *Стивенс У. Р.* UNIX. Процестердің өзара әрекеттесуі / У Р. Стивенс. — СПб. : Питер, 2002. — 576 б.

10. *Стивенс У. Р.* UNIX. Желілік қосымшаларды әзірлеу / У. Р. Стивенс. — СПб. : Питер, 2003. — 1088 б.

11. *Строкин Г.* BASH-конспект / Г. Строкин // http://www.linux.org.ru/ books/bash-conspect.html.

12. *Таненбаум Э.* Заманауи операциялық жүйелер / Э. Таненбаум. — СПб. : Питер, 2002. — 1040 б.

13. *Фридл Дж.* Реттелетін белгілер. Бағдарламалаушының кітапханасы / Дж. Фридл. — СПб. : Питер, 2001. — 352 б.

14. *Хендриксен Д.* UNIX және Windows NT біріктіруі / Д. Хендриксен. — М. : Диа-Софт, 1999. — 352 б.

15. *Цилюрик О.* QNX/UNIX. Параллелизмнің анатомиясы / О. Цилюрик, Е. Горошко. — М. : Символ-Плюс, 2006. — 288 б.

16. *Чан Т.* UNIX арналған C++ жүйелік бағдарламалау / Т. Чан. — Киев. : BHV, 1997. — 592 с.

17. *Шнитман В. З.* Кооперативті ақпараттық жүйелердің аппартты-бағдарламалық платформалары / В. З. Шнитман, С. Д. Кузнецов. // http://www.citforum.ru/hardware/app_kis/contents.shtml.

18. *Dijkstra E. W.* Cooperating Sequential Processes, in Programming Languages / Ed. F. Genuys. — Academic Press, 1968. — P. 43 — 112.

19. *Maurice J. Bach.* The Design of the UNIX Operating System. — Prentice - Hall, Inc., Englewood Cliffs, New Jersey, 1986. — 471 p.

Мазмұны

Оқу жарияланымы

Батаев Алексей Владимирович,
Налютин Никита Юрьевич,
Синицын Сергей Владимирович

Операциялық жүйелер және орталар

Оқулық

Редактор *Е. А. Тульсанова*
Техникалық редактор *Е. Ф. Коржуева*
Компьютерлік беттеуші: *Д. В. Федотов*
Түзетуші *И. А. Ермакова*

Жарияланым № 101116318. Мөрге қол қойылған 30.12.2013. Форматы 60 х 90/16.
Гарнитура «Baltica». Офсетті парақ № 1. Офсетті мөр. Шартта тасп.беті 17,0.
Тираж 1 500 дана. Тапсырыс №

«Академия» ААҚ баспа орталығы. www.academia-moscow.ru 125252,
Мәскеу, Мира даңғылы, 101В, б. 1, а/я 48.
Тел./факс: (495)648-0507, 616-0029.

Санитарлық-эпидемиологиялық тұжырым № РОСС RU.AE51.Н16476. 05.04.2013.

«Идеал-Пресс» басылып шығарылды.