ҚАРАҒАНДЫ ТЕХНИКАЛЫҚ УНИВЕРСИТЕТІ

Ф.М. БАРТОСИК, Д.Т. КОЖАНОВА, Л.Б. КАДЫРОВА

ОПЕРАЦИЯЛЫҚ ЖҮЙЕЛЕР ҚАУІПСІЗДІГІ

Қарағанды 2021

ҚАРАҒАНДЫ ТЕХНИКАЛЫҚ УНИВЕРСИТЕТІ

«Ақпараттық технологиялар және қауіпсіздік» кафедрасы

Ф.М. БАРТОСИК, Д.Т. КОЖАНОВА, Л.Б. КАДЫРОВА

ОПЕРАЦИЯЛЫҚ ЖҮЙЕЛЕР ҚАУІПСІЗДІГІ

Университеттің Ғылыми кеңесі
оқу құралы ретінде бекіткен

Қарағанды 2021

ӘОЖ 004.451(075)
КБЖ 32.973.26-018.2я73
Б24

Университеттің Редакциялық-баспа кеңесі ұсынған

Пікір жазғандар:

Көккөз М.М., педагогика ғылымдарының кандидаты, доцент, ҚарТУ АТҚ кафедрасының меңгерушісі;

Казимова Д.А., п.ғ.к, доцент, Е.А. Бөкетов атындағы ҚарУ математика және ақпараттық технологиялар факультетінің деканы;

Есенбаев С.Х., т.ғ.к., ҚарТУ- нің Редакциялық - баспа кеңесінің мүшесі, АТҚ кафедрасының доценті

Бартосик Ф.М.

Б24 Операциялық жүйелер қауіпсіздігі: Оқу құралы/Ф.М. Бартосик, Д.Т. Кожанова, Л.Б. Кадырова; Қарағанды техникалық университеті. -Қарағанды: ҚарТУ баспасы, 2021. -77б.

ISBN 978-601-320-548-9

Оқу құралы оқу жоспарының талаптарына және «Операциялық жүйелер қауіпсіздігі» пәнінің бағдарламасына сәйкес құрастырылған.

Жүйелік бағдарламалау мәселелері, операциялық жүйелердің түрлері, мақсаты мен функциялары қарастырылады. Операциялық жүйелердің тарихы мен даму тенденциялары, операциялық жүйелердің негізгі тұжырымдамалары егжей-тегжейлі сипатталған. Ағындар мен процестерді басқару, соның ішінде оларды басқару және жоспарлау негіздері сипатталған. Маңыздылығы үнемі өсіп келе жатқан көп ядролы және көп ядролы жүйелерге; виртуалды жадты басқаруға; файлдық жүйенің құрылымы мен басқаруына көп көңіл бөлінеді. Операциялық жүйелердің қауіпсіздігіне төнетін қауіптер мен қорғау тәсілдері сипатталады; белгілі операциялық жүйелердегі объектілердің қауіпсіздігін басқарудың негізгі аспектілері: Linux, Android, Windows.

Оқу құралы 6В06301 «Ақпараттық қауіпсіздік жүйелері» білім беру бағдарламасының студенттеріне арналған және пәнаралық мамандықтар студенттеріне де пайдалы болуы мүмкін.

ӘОЖ 004.451(075)
КБЖ 32.973.26-018.2я73

ISBN 978-601-320-548-9

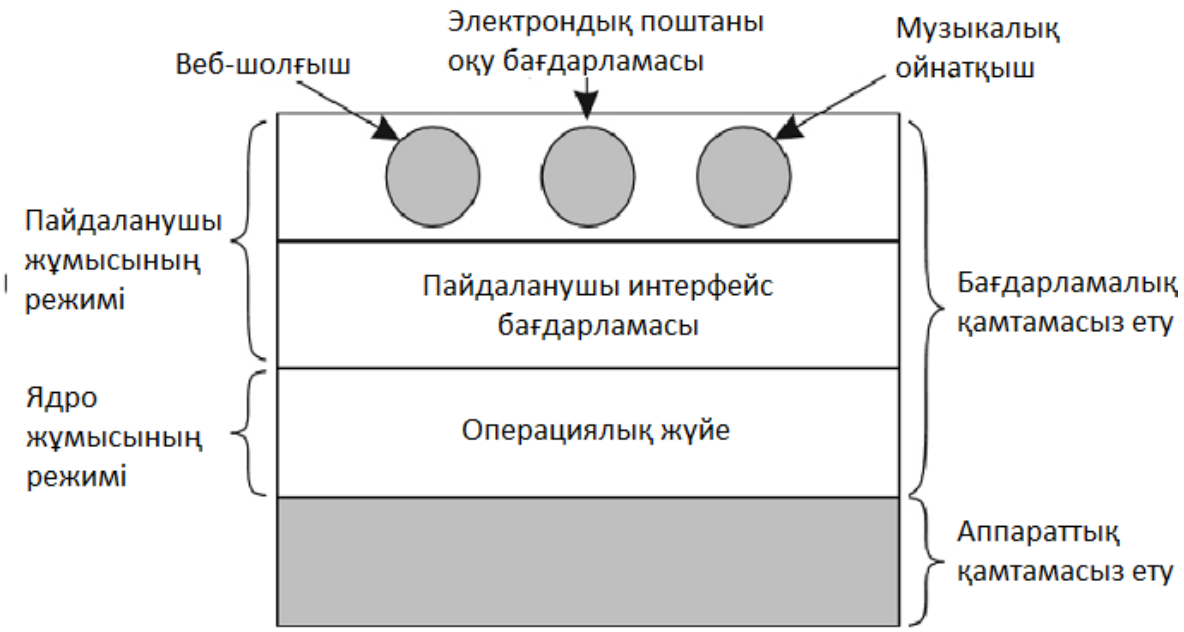
МАЗМҰНЫ

КІРІСПЕ

Операциялық жүйенің мақсаты

Қазіргі компьютер бір немесе бірнеше процессорлардан, жедел жадынан, дискілерден, принтерден, пернетақтадан, тышқаннан, дисплейден, желілік интерфейстерден және басқа да енгізу / шығару құрылғыларынан тұрады. Нәтижесінде күрделі жүйе пайда болады. Егер қолданбалы бағдарламаны жасайтын әр бағдарламашыға осы құрылғылардың барлық жұмысының қыр-сырын түсіну қажет болса, онда ол кодтың бір жолын жазбайды. Сонымен қатар, осы компоненттердің барлығын басқару және оларды оңтайлы пайдалану өте қиын мәселе. Осы себептен компьютерлерде операциялық жүйе деп аталатын бағдарламалық жасақтаманың арнайы қабаты жабдықталған, оның міндеті пайдаланушы бағдарламаларын, сонымен қатар, бұрын аталған ресурстардың барлығын басқару болып табылады.

Операциялық жүйенің орналасуы 1-суретте көрсетілген. Ол тікелей аппараттық құралдармен жұмыс істейді және қалған бағдарламалық жасақтаманың негізі болып табылады.



1-сурет - Бағдарламалық қамтамасыз ету құрылымындағы операциялық жүйенің орны

Суреттің төменгі жағында жабдық көрсетілген. Ол микросхемалардан, тақталардан, дискілерден, пернетақтадан, монитордан және басқа физикалық

объектілерден тұрады. Аппараттық құралдың үстінде бағдарламалық жасақтама орналасқан. Көптеген компьютерлерде екі режим жұмыс істейді: ядрорежимі және пайдаланушы режимі. Операциялық жүйе - ядро режимінде жұмыс жасайтын бағдарламалық жасақтаманың ең негізгі бөлігі (супервайзер режимі деп те аталады). Бұл режимде ол барлық жабдықтарға толық қол жеткізе алады және машина орындай алатын кез келген нұсқауды орындай алады. Бағдарламалық жасақтаманың қалған бөлігі пайдаланушының режимінде жұмыс істейді, онда машинаның нұсқауларының бір бөлігі ғана болады. Атап айтқанда, пайдаланушы режимінде жұмыс істейтін бағдарламаларға машинаны басқаратын немесе енгізу / шығару (енгізу-шығару) операцияларын орындайтын нұсқауларды пайдалануға тыйым салынады. Біз осы кітаптың беттерінде ядро мен пайдаланушы режимдерінің арасындағы айырмашылықтарға бірнеше рет ораламыз. Бұл айырмашылықтар операциялық жүйенің жұмысына шешуші әсер етеді.

Пайдаланушы интерфейсінің бағдарламалары - қабық немесе GUI - қолданушы режиміндегі бағдарламалық жасақтаманың ең төменгі деңгейінде және пайдаланушыға веб-браузер, электрондық пошта оқу құралы немесе музыкалық ойнатқыш сияқты басқа бағдарламаларды іске қосуға мүмкіндік береді. Бұл бағдарламалар операциялық жүйені де белсенді қолданады.

Көптеген жүйелерде операциялық жүйелерге көмектесетін немесе арнайы функцияларды орындайтын қолданушы режиміндегі бағдарламалар да бар. Мысалы, пайдаланушыларға парольдерін өзгертуге мүмкіндік беретін бағдарламалар жиі кездеседі. Олар операциялық жүйеге кірмейді және ядро режимінде жұмыс істемейді, бірақ бәрі олардың маңызды функцияны орындайтынын және арнайы қорғалуы керек екенін түсінеді. Кейбір жүйелер бұл идеяны экстремалды формаға жеткізеді, ал дәстүрлі түрде операциялық жүйеге жататын аймақтар (мысалы, файлдық жүйе) пайдаланушы кеңістігінде жұмыс істейді. Мұндай жүйелерде нақты сызық жүргізу қиын. Ядролық режимде жұмыс жасайтын барлық бағдарламалар, әрине, амалдық жүйенің бөлігі болып табылады, бірақ ядро режимінен тыс жұмыс істейтін кейбір бағдарламалар оның бөлігі болуы да мүмкін немесе онымен тығыз байланыста болады. Операциялық жүйелер қолданушы бағдарламаларынан (яғни қосымшалардан) тек орналасуымен ерекшеленеді. Олардың ерекшеліктері өте үлкен көлем, күрделі құрылым және ұзақ мерзімді пайдалану. Linux немесе Windows сияқты операциялық жүйенің бастапқы коды шамамен 5 миллион жолды құрайды. Осы көлемді елестету үшін 5 миллион жолды кітап форматында бір параққа 50 жолдан және әр томға 1000 парақтан ойша басып шығарайық (бұл осы кітаптан көп). Іс жүзінде тұтас кітап сөресі болып табылатын бұл кодты басып шығару үшін 100 том қажет болады. Сіз операциялық жүйені қолдау туралы тапсырма алдыңыз деп

елестете аласыз және бірінші күні бастығыңыз сізді кітап сөресіне апарып: «Мұның бәрін білу керек» деді. Және бұл тек ядро режимінде жұмыс жасайтын бөлікке қатысты. Қажетті ортақ кітапханаларды қоса, Windows 70 миллион кодтық кодтан асады (қағазға басылған, олар 10-20 кітап сөрелерін алады) және бұл негізгі қосымшаларды есептемейді (мысалы, Windows Explorer, Windows Media Player және т.б.).

Операциялық жүйе ресурстар менеджері ретінде

Операциялық жүйе негізінен қолданбалы бағдарламалар үшін абстракцияны ұсынады деген идея жоғарыдан төменге қарау. Балама көзқарасты жақтаушылар, төменнен жоғарыға қарай, күрделі жүйенің барлық бөліктерін басқару үшін операциялық жүйе бар деген пікірді ұстанады. Қазіргі компьютерлер процессорлардан, жадыдан, таймерлерден, дискілерден, тышқандардан, желілік интерфейстерден, принтерлерден және басқа да көптеген құрылғылардан тұрады. Төменнен жоғары қарай қарауды жақтаушылар операциялық жүйенің міндеті оларды қолдануды талап ететін әртүрлі бағдарламалар арасында процессорларды, жадты және енгізу-шығару құрылғыларын реттелген және басқарылатын бөлуді қамтамасыз ету деп санайды. Қазіргі заманғы операциялық жүйелер бірнеше бағдарламалардың бір уақытта жұмыс істеуіне мүмкіндік береді. Егер бір компьютерде жұмыс істейтін барлық үш бағдарлама өз нәтижелерін бір уақытта бір принтерде басып шығаруға тырысса, не болатынын елестетіп көріңіз. Басып шығарудың алғашқы бірнеше жолдары № 1 бағдарламадан, келесі бірнеше жолдар № 2 бағдарламадан, содан кейін № 3 бағдарламадан бірнеше жолдар және т.б. болуы мүмкін. Амалдық жүйе принтерге арналған барлық шығыс деректерін дискіде Буферлеу арқылы ықтимал хаосты тазартуға арналған. Бір бағдарлама жұмысын аяқтағаннан кейін, операциялық жүйе оның шығысын олар сақталған дискідегі файлдан принтерге көшіре алады, сонымен бірге, басқа бағдарлама деректерді шығаруды жалғастыра алады, бұл нәтиже принтерге нақты (уақытқа дейін) түспейтінін байқамайды. Компьютермен (немесе желімен) бірнеше пайдаланушылар жұмыс істеген кезде, жадты, енгізу-шығару құрылғыларын және басқа ресурстарды басқару және қорғау қажеттілігі айтарлықтай артады, өйткені пайдаланушылар бір-бірінің жұмысына кедергі келтіреді. Сонымен қатар, пайдаланушылар көбінесе аппараттық құралдарды ғана емес, сонымен қатар ақпаратты (файлдар, мәліметтер базасы және т.б.) бөлісуі керек. Қысқаша айтқанда, операциялық жүйеге осы көзқарастың жақтаушылары оның негізгі міндеті ресурстарды пайдалану сұраныстарын қанағаттандыру, оларды пайдалану үшін

жауапкершілік алу және т.б. үшін қандай бағдарлама қолданылатынын бақылау деп санайды.

Ресурстарды басқару ресурстарды екі түрлі жолмен: уақыт пен кеңістікте мультиплекстеуді (бөлуді) қамтиды. Ресурс уақыт өте келе бөлінген кезде, әртүрлі бағдарламалар немесе пайдаланушылар оны кезекпен пайдаланады: алдымен ресурс кейбіреулерге, содан кейін басқаларға және т.б. Мысалы, тек бір орталық процессорға және оны орындауға тырысатын бірнеше бағдарламаларға ие бола отырып, Операциялық жүйе алдымен орталық процессорды бір бағдарламаға бөледі, содан кейін ол жеткілікті жұмыс істегеннен кейін орталық процессор басқа бағдарламаны, содан кейін басқа бағдарламаны өз қолына алады, және, ақырында, бірінші бағдарлама оны қайтадан өз қолына алады. Ресурстың уақытында қалай бөлінетінін анықтау - келесі тұтынушы кім болады және қанша уақыт операциялық жүйенің міндеті.

Уақытты мультиплекстеудің тағы бір мысалы - принтерді бөлісу. Бір принтерде басып шығару үшін кезекте бірнеше басып шығару тапсырмасы болған кезде, қайсысы келесіде орындалатынын шешу керек. Ресурстарды бөлудің тағы бір түрі-кеңістікті бөлу. Кезек-кезек жұмыс істеудің орнына әр клиент ортақ ресурстың бір бөлігін алады. Мысалы, жедел жад әдетте бірнеше жұмыс істейтін бағдарламалар арасында бөлінеді, сондықтан олардың барлығы бір уақытта жадта болуы мүмкін (мысалы, орталық процессорды кезекпен пайдалану). Жад бірнеше бағдарламаны сақтау үшін жеткілікті болған жағдайда, бір бағдарламаға бүкіл жадты бөлуден гөрі бірден бірнеше бағдарламаны жадқа орналастырған тиімді, әсіресе егер ол жалпы кеңістіктің аз ғана бөлігін қажет етсе. Әрине, тең қолжетімділік, қауіпсіздік және т.б. проблемалар туындайды және операциялық жүйе оларды шешуі керек. Бөлінген кеңістігі бар тағы бір ресурс - қатты диск. Бір дискідегі көптеген жүйелерде көптеген пайдаланушыларға тиесілі файлдарды бір уақытта сақтауға болады. Дискілік кеңістікті бөлу және дискілік блоктарды кім қолданатынын бақылау - бұл операциялық жүйенің ресурстарды басқаруға арналған әдеттегі міндеті.

1 ПӘНДІ ОҚУ МІНДЕТТЕРІ

1.1 Бірінші буын (1945-1955): электронды шамдар

Баббидждің сәтсіз талпыныстарынан кейін екінші дүниежүзілік соғысқа дейін цифрлық компьютерлерді жобалаудағы прогресс іс жүзінде байқалмады, бұл олармен жұмыс істеудің жарылғыш белсенділігін арттырды. Профессор Джон Атанасов (John Atanasoff) және оның аспиранты Клиффорд Берри (Clifford Berry) Айова мемлекеттік университетінде қазіргі уақытта алғашқы сандық компьютер болып саналатын дизайн жасады. Онда 300 электронды шам қолданылды. Осы уақытта Берлинде Конрад Зузе (Konrad Zuse) электромеханикалық релелерді қолдануға негізделген Z3 компьютерін салды. 1944 жылы ғалымдар тобы (оның ішінде Алан Тюринг) Блетшли паркінде (Ұлыбритания) Колоссус салынды және бағдарламаланды, Гарвардта Ховард Айкен (Howard Aiken) "Марк I", ал Пенсильвания мемлекеттік университетінде Уильям Мочли (Уильям Маучли) және оның аспиранты Джон Преспер Эккерт (J.Presper Eckert) "Эниак" салды. Бұл машиналардың кейбіреулері сандық болды, басқаларында электронды шамдар қолданылды, ал олардың бір бөлігінің жұмысын бағдарламалауға болады, бірақ олардың бәрі өте қарапайым болды және тіпті қарапайым есептеулерді жасауға бірнеше секунд жұмсады.

Компьютерлік дәуірдің басында әр машинаны сол адамдар тобы (әдетте инженерлер) жобалаған, жасаған, бағдарламалаған, пайдаланған және қызмет еткен. Барлық бағдарламалау тек машина тілінде жүргізілді, тіпті одан да жаманы, электр тізбектерін құрастырудың арқасында және машинаның негізгі функцияларын басқару үшін мыңдаған сымдарды коммутациялық панельдерге қосуға тура келді. Ол кезде бағдарламалау тілдері туралы (тіпті ассемблер туралы) әлі ештеңе білмеген. Операциялық жүйелер туралы ешкім ештеңе естіген жоқ. Бағдарламашының жұмыс режимі арнайы стендте белгілі бір машина уақытына жазылу, содан кейін машина залына түсу, коммутациялық панельді компьютерге салу және келесі бірнеше сағатты жұмыс барысында 20 мыңға жуық электронды шамдардың ешқайсысы істен шықпайды деген үмітпен өткізу болды. Шын мәнінде, барлық шешілетін есептер синус, косинус және логарифм кестелерін нақтылау немесе артиллериялық снарядтардың ұшу жолдарын есептеу сияқты қарапайым математикалық және сандық есептеулерге дейін азайтылды.

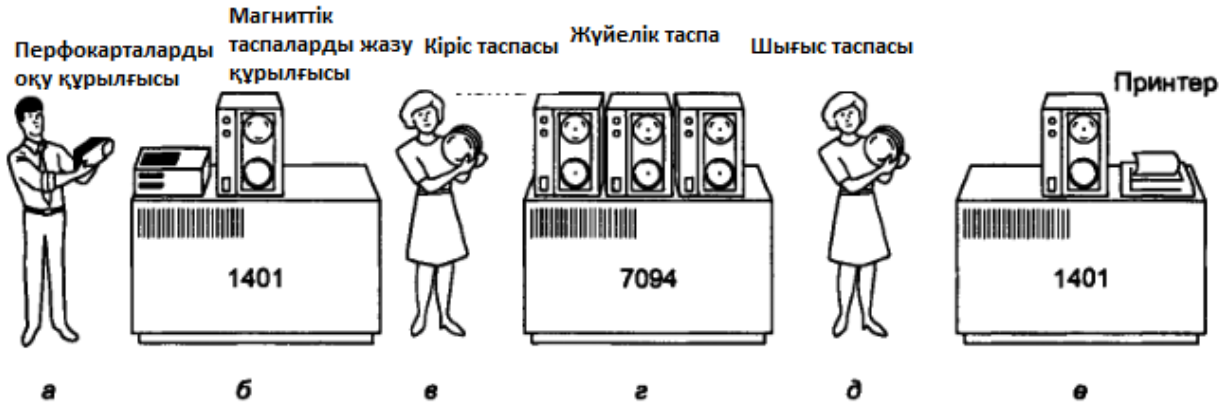
1950 жылдардың басында карточкалар пайда болған кезде жағдай біршама жақсарды. Коммутациялық панельдерді қолданудың орнына, бағдарламаларды карталарға жазып, олардан оқуға болады, бірақ әйтпесе жұмыс процедурасы өзгерген жоқ.

1.2 Екінші буын (1955-1965): транзисторлар мен пакеттік өңдеу жүйелері

1950 жылдардың ортасында транзисторларды ойлап табу және қолдану бүкіл суретті түбегейлі өзгертті. Компьютерлер жеткілікті сенімді бола бастады, пайдалы функцияларды орындай отырып, машиналардың ұзақ уақыт жұмыс істеу ықтималдығы жоғары болды. Алғаш рет дизайнерлер, құрастырушылар, операторлар, бағдарламашылар және техникалық қызмет көрсету қызметкерлері арасында нақты бөліну болды.

Қазір негізгі кадрлар деп аталатын машиналар арнайы ауа баптағыш бөлмелерде орналасқан, онда оларды кәсіби операторлардың бүкіл құрамы басқарған.

Пакеттік өңдеу жүйесі жалпы қабылданған шешім болды. Бастапқыда жоспар кіріс бөлмесінде тапсырмалардың толық науасын (палубалық карточкалар) жинап, содан кейін оларды IBM 1401 сияқты кішкентай және (салыстырмалы түрде) арзан компьютерді қолдана отырып, магниттік таспаға қайта жазу болды, ол карталарды оқу, таспаларды көшіру және шығыс деректерін басып шығару үшін өте жақсы болды, бірақ сандық есептеулерге сәйкес келмеді. IBM 7094 сияқты басқа да қымбат машиналар нақты есептеулер үшін пайдаланылды. Бұл процесс 2-суретте көрсетілген.



2-сурет – Пакеттік өңдеудің ертедегі жүйесі

Бағдарламашы IBM 1401 карталарын әкеледі (2, а-сурет). IBM 1401 тапсырмалар партиясын магниттік таспаға жазады (2, б-сурет). Оператор таспаға енгізілген деректерді IBM 7094-ке жібереді (2, с-сурет). IBM 7094 есептеулерді орындайды (2, г-сурет). Оператор таспаны шығыс деректерімен IBM 1401-ге жібереді (2, д-сурет). IBM 1401 шығаруды басып шығарады (2, е-сурет).

Пакетті жинағаннан кейін шамамен бір сағат өткеннен кейін карточкалар магниттік лентаға оқылды, оны машиналық бөлмеге алып барды да, оларды ленталық жинақтағышқа қойды. Содан кейін оператор арнайы жұмысты жүктеді (бүгінгі операциялық жүйенің прототипі), ол бірінші жұмысты таспадан оқып шығады және оны іске қосады. Шығарма басудың орнына екінші таспаға жазылды. Келесі тапсырманы орындағаннан кейін, амалдық жүйе таспадан келесіні автоматты түрде оқып, оны өңдей бастады. Бүкіл пакетті өңдеп болғаннан кейін, оператор кіріс және шығыс ақпараттары бар таспаларды алып тастап, келесі тапсырмасы бар жаңа таспа салып, дайын деректерді офлайн режимде басып шығару үшін IBM 1401-ге орналастырды (яғни, хостпен байланыссыз) компьютер).

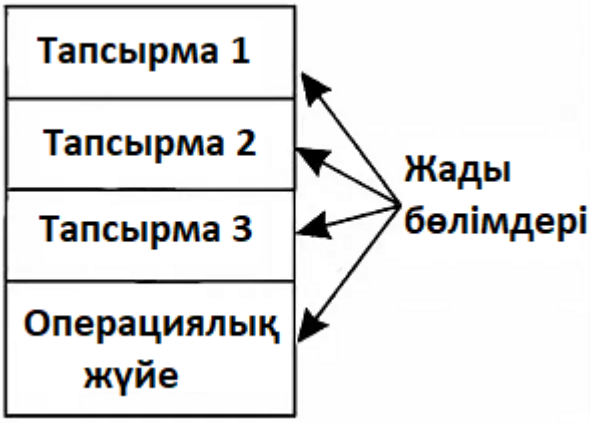
1.3 Үшінші буын (1965-1980 жж.): интегралды микросхемалар және көптапсырмалық

1960 жылдардың басында компьютер өндірушілерінің көпшілігінде екі түрлі, үйлесімсіз отбасылар пайда болды. Бір жағынан, бұл IBM 7094 сияқты сөзді өңдейтін ғылыми компьютерлер еді, олар ғылым мен техникадағы өнеркәсіптік деңгейдегі сандық есептеулер үшін пайдаланылды, екінші жағынан, белгілерді сипаттамалық өңдейтін коммерциялық компьютерлер, мысалы деректер мен тапсырмаларды сұрыптау және басып шығару үшін банктер мен сақтандыру компаниялары кеңінен қолданатын IBM 1401.

IBM бұл проблемаларды IBM System / 360 сериялы машиналармен бір уақытта шешуге тырысты. Бұл компьютерлер мөлшері бойынша салыстырылатын машиналардан IBM 1401-ге дейін IBM 7094-тен едәуір үлкен және қуатты машиналарға дейін болатын бағдарламалық жасақтамаға сәйкес машиналардың сериясы болды. Бұл компьютерлер тек бағасы мен өнімділігімен ерекшеленді (жады, процессордың максималды жылдамдығы, мүмкін енгізу-шығару құрылғыларының саны және т.б.). Барлық машиналардың құрылымы мен командалар жиынтығы бірдей болғандықтан, бір компьютерге арналған бағдарламалар, басқаларында, ең болмағанда теория жүзінде жұмыс істей алады.

IBM/360 компьютерлер отбасылығы жеке транзисторлары бар екінші буын машиналарына қарағанда баға мен сапаның артықшылығын ұсынатын шағын интегралды микросхемаларды қолданудың алғашқы ірі сериясы болды. OS / 360 және басқа компьютер өндірушілерінің осыған ұқсас үшінші буын операциялық жүйелері оның үлкен көлеміне және кемшіліктеріне қарамастан, көптеген клиенттердің қажеттіліктерін қанағаттандырды. Олар тіпті екінші буын операциялық жүйелерінде жоқ бірнеше негізгі техниканы танымал етті. Ең маңызды жетістік - көптапсырмалық.

IBM 7094 компьютерінде ағымдағы жұмыс магниттік таспадан немесе басқа құрылғылардан кіріс-шығыс операцияларын күту кезінде тоқтатылған кезде, Орталық процессор кіріс-шығыс жұмысы аяқталғанға дейін тоқтап қалды. Күрделі ғылыми есептеулерде I / O құрылғылары аз қарқынды жұмыс істеді, сондықтан күтуге кететін уақыт онша маңызды болмады. Бірақ деректерді коммерциялық өңдеу кезінде і / о құрылғысының күту уақыты бүкіл жұмыс уақытының 80 немесе 90% алуы мүмкін, сондықтан өте қымбат процессордың ұзақ жұмыс уақытын жою үшін бір нәрсе жасау керек болды. Бұл мәселені шешу жадыны бөлімдер деп аталатын бірнеше бөлікке бөлу болды, олардың әрқайсысында жеке тапсырма орындалды (3-сурет). Бір тапсырма енгізу-шығару құрылғысының аяқталуын күтіп тұрған кезде, екіншісі Орталық процессорды қолдана алады. Егер жедел жадта жеткілікті тапсырмалар болса, орталық процессорды уақыттың барлық 100% дерлік жүктеуге болады. Бір уақытта жадта сақталатын көптеген тапсырмалар әр тапсырманы жад аймақтарын заңсыз тағайындаудан және басқа тапсырмаларға зиян келтіруден қорғау үшін арнайы жабдықты қажет етті. Осы мақсатта 360 сериялы Компьютерлер және басқа үшінші буын жүйелері арнайы жабдықтармен жабдықталған.



3-сурет - Жадыда үш тапсырмасы бар көптапсырмалық жүйесі

Үшінші буынның операциялық жүйелерінің тағы бір маңызды артықшылығы - компьютерлік бөлмеге перфокарталардан дискіге дейінгі жұмысты оқи алу. Әрбір ағымдағы тапсырманың соңында амалдық жүйе дискіден босатылған жад бөліміне жаңа тапсырма жүктей алады және осы тапсырманы бастай алады. Бұл әдіс деректерді ауыстыру немесе сполинг деп аталады. Жауапты күту уақытын қысқартуға деген ұмтылыс уақытты бөлісу режимін дамытуға әкелді - әр пайдаланушының өзінің диалогтық терминалы болатын көп тапсырма нұсқасы.

1.4 Төртінші буын (1980 жылдан осы күнге дейін): дербес компьютерлер

Операциялық жүйелер эволюциясының келесі кезеңі ҮИС - үлкен интегралды схемалардың - бір шаршы сантиметрде мыңдаған транзисторлары бар кремний чиптерінің пайда болуымен байланысты. 1980 жылдардың басында IBM корпорациясы IBM PC жасап, оған бағдарламалық жасақтаманы іздей бастады. IBM қызметкерлері Билл Гейтспен байланысып, оның негізгі тіл аудармашысын пайдалану құқығына лицензия алды. Сондай-ақ, олар IBM PC-де жұмыс істейтін операциялық жүйені білмейді ме деп сұрады. Гейтс Операциялық жүйелер саласындағы жетекші компания болған Digital Research компаниясына жүгінуге кеңес берді. Бірақ Килделл IBM-мен кездесуден бас тартып, оның орнына бағыныштысын жіберді. Ең жаманы, оның адвокаты істі толығымен бұзғаннан гөрі IBM PC-тің әлі шығарылмағаны туралы жарияламау туралы келісімге қол қоюдан бас тартты. IBM тағы да Гейтстен оны операциялық жүйемен қамтамасыз етуді сұрады. Қайта өңдеуден кейін Гейтс жергілікті Компьютер өндірушісі Seattle Computer Products - тің сәйкес DOS операциялық жүйесі (Disk Operating System-дисктік операциялық жүйе) бар екенін анықтады. Ол бұл компанияға Seattle Computer Products компаниясы қабылдаған DOS (шамамен $50,000) сатып алу ұсынысымен бет алды. Содан кейін Гейтс DOS/BASIC бағдарламалық пакетін жасады және пакетті IBM сатып алды. Өзгертілген жүйе MS-DOS (MicroSoft Disk Operating System) деп аталды және IBM PC нарығында тез үстем болды. Ең бастысы, Гейтстің MS-DOS-ты компьютерлік компанияларға Килделлдің CP/M-ді соңғы пайдаланушыларға (кем дегенде бастапқы кезеңде) сатуға тырысуынан айырмашылығы, олардың жабдықтарымен бірге орнату үшін сату туралы шешімі болды.

CP/M, MS-DOS және алғашқы микрокомпьютерлерге арналған басқа операциялық жүйелер пайдаланушының пернетақтадан енгізген командаларына негізделген. Уақыт өте келе, 1960 жылдары Стэнфорд ғылыми-зерттеу институтында (Stanford Research Institute) Даг Энгельбарт (Douglas Engelbart) жүргізген зерттеулердің арқасында жағдай өзгерді. Энгельбарт терезелермен, белгішелермен, мәзір жүйелерімен және тінтуірмен бірге графикалық пайдаланушы интерфейсін (GUI, графикалық пайдаланушы интерфейсі) ойлап тапты. Бұл идеяны Xerox PARC зерттеушілері қабылдады және олар жасаған машиналарда қолданды.

Бір күні Стив Джобс (Steve Jobs), гаражында құрылған Apple компьютерінің авторларының бірі PARC-ке барып, онда GUI-ді көріп, Xerox басшылығы бағаламайтын әлеуеттің деңгейін бірден түсінді. Содан кейін Джобс графикалық пайдаланушы интерфейсімен жабдықталған Apple

компьютерін құруға кірісті. Бұл жоба Lisa компьютерін құруға әкелді, ол өте қымбат және коммерциялық жетістікке жете алмады. Джобстың екінші әрекеті, Apple Macintosh компьютері, Lisa-ға қарағанда едәуір арзан болғандықтан ғана емес, сонымен қатар компьютерлерде бұзылмаған және бір нәрсені үйренуге мүлдем тырыспайтын пайдаланушыларға арналған пайдаланушыға ыңғайлы интерфейс болғандықтан үлкен жетістікке жетті. Macintosh компьютерлері шығармашылық кәсіптердің өкілдері - суретші-дизайнерлер, кәсіби сандық фотографтар және кәсіби сандық бейне өнімдерін өндірушілер арасында кеңінен қолданылады, оларды үлкен ықыласпен қабылдады. 1999 жылы Apple компаниясы BSD UNIX ядросын ауыстыру үшін Карнеги Меллон Университетінің мамандары жасаған Mach микро ядросынан шыққан ядродан қарыз алды. Сондықтан, Mac OS X - бұл UNIX негізінде жасалған операциялық жүйе, бірақ өте ерекше интерфейсі бар.

Microsoft корпорациясы MS-DOS мұрагерін құруға шешім қабылдаған кезде, Ол Macintosh-тың сәттілігінен үлкен әсер алды. Нәтижесінде, Windows деп аталатын графикалық пайдаланушы интерфейсін қолдануға негізделген жүйе пайда болды, ол бастапқыда MS-DOS үстінен қондырма болды (яғни, ол нақты операциялық жүйеге қарағанда қабыққа ұқсас болды). Шамамен 10 жыл бойы, 1985 жылдан 1995 жылға дейін, Windows MS-DOS үстінде жұмыс істейтін графикалық қабық болды. Алайда, 1995 жылы Windows - тың тәуелсіз нұсқасы шығарылды-Windows 95. Ол операциялық жүйенің көптеген функцияларын тек жүктеу үшін, сондай-ақ MS-DOS үшін жасалған ескі бағдарламаларды орындау үшін өзінің құрамына кіретін MS-DOS жүйесін қолдана отырып орындады. 1998 жылы Windows 98 деп аталатын осы жүйенің сәл өзгертілген нұсқасы шығарылды. Алайда, бұл екі жүйеде де, Windows 95 және Windows 98-де де 16 биттік Intel процессорлары үшін Ассемблерде жазылған кодтың жеткілікті мөлшері бар.

Microsoft корпорациясының тағы бір операциялық жүйесі Windows NT (NT New Technology — жаңа технология дегенді білдіреді) болды, ол белгілі бір деңгейде Windows 95-пен үйлесімді. Алайда, ол қайтадан жазылды және толыққанды 32 биттік жүйе болды. Windows NT-тің жетекші әзірлеушісі Дэвид Катлер болды, ол VAX VMS операциялық жүйесін жасаушылардың бірі болды, сондықтан VMS-тің кейбір идеялары NT-де де бар (және VMS иесі, DEC, Microsoft корпорациясын сотқа берді. Жанжал соттан тыс тәртіппен лайықты сомаға шешілді). Microsoft бірінші нұсқасы MS-DOS және Windows-тың барлық басқа нұсқаларын ауыстырады деп күтті, өйткені ол олардан әлдеқайда жоғары болды, бірақ үміт орындалмады. Тек Windows NT 4.0 операциялық жүйесі, әсіресе корпоративті желілерде танымал бола алды. Windows NT-нің бесінші нұсқасы 1999 жылдың басында Windows 2000 деп

аталды. Ол екі нұсқаны да ауыстыруға арналған - Windows 98 және Windows NT 4.

1.5 Бесінші буын (1990 жылдан қазіргі уақытқа дейін): мобильді компьютерлер

Алғашқы шынайы ұялы телефон 1946 жылы пайда болды, содан кейін оның салмағы шамамен 40 кг болды. Оны тек автокөлік көмегімен тасымалдауға болатын еді. Алғашқы шынайы портативті телефон 1970 жылдары пайда болды және салмағы шамамен 1 кг өте оң қабылданды. Оны «кірпіш» деп атады. Көп ұзамай мұндай құрылғыға деген ұмтылыс жалпыға ортақ болды. Қазіргі уақытта әлем халқының 90%-ы ұялы байланысты пайдаланады. Жақында ұялы телефондар мен қол сағаттарынан ғана емес, көзілдіріктер мен басқа да киілетін заттардан қоңырау шалуға болады. Сонымен қатар, телефонға тікелей қатысы бар құрылғының бөлігі енді ешқандай қызығушылық тудырмайды. Бұл туралы ойланбастан біз электрондық поштаны аламыз, веб-парақтарды қарап шығамыз, достарымызға мәтіндік хабарлар жібереміз, ойын ойнаймыз және көшелерде кептелістердің бар-жоғын білеміз.

Бір құрылғыда телефонды және компьютерді біріктіру идеясы 1970-ші жылдардан бастап пайда болғанына қарамастан, алғашқы нақты смартфон 1990-шы жылдардың ортасында, Nokia өзінің N9000-ды шығарған кезде пайда болды, ол екі бөлек құрылғының: телефон мен PDA-нің тіркесімі болды. 1997 жылы Ericsson компаниясында оның gs88 «Penelope» өнімі үшін «смартфон» термині ойлап табылды.

Енді смартфондар кеңінен таралған кезде, әртүрлі операциялық жүйелер арасында қатаң бәсекелестік орнады, оның нәтижесі дербес компьютерлер әлеміне қарағанда одан да айқын емес. Бұл жолдарды жазу кезінде Google Android операциялық жүйесі басым болды, ал екінші орында Apple iOS болды, бірақ келесі бірнеше жылда жағдай өзгеруі мүмкін. Смартфондар әлемінде бір ғана нәрсе анық: ұзақ уақыт бойы операциялық жүйелердің кез-келгенінде қалу өте қиын болады.

Пайда болғаннан кейінгі алғашқы онжылдықта смартфондардың көпшілігі Symbian OS-мен жұмыс істеді. Бұл операциялық жүйені Samsung, Sony Ericsson, Motorola және Nokia сияқты танымал брендтер таңдады. Бірақ Symbian нарығының үлесі RIM Blackberry OS (2002 жылы смартфондар үшін шығарылған) және Apple iOS (2007 жылы алғашқы iPhone үшін шығарылған) сияқты басқа операциялық жүйелерді таңдай бастады. Көптеген адамдар rim бизнес-құрылғы нарығында басым болады деп күтті iOS тұтынушылық құрылғылар нарығын жеңеді. Нарық үшін Symbian танымалдығы төмендеді.

2011 жылы Nokia Symbian-дан бас тартып, Windows Phone-ға назар аударудың негізгі платформасы ретінде өзінің ниетін жариялады. Біраз уақыт бойы Apple және RIM операциялық жүйелері барлығына ұнады (олар Symbian - мен бірдей үстем позицияға ие болмаса да), бірақ көп ұзамай олардың барлық қарсыластары Linux ядросына негізделген Android операциялық жүйесін басып озды, оны 2008 жылы Google іске қосты.

Телефон өндірушілері үшін Android-тің ашық көзі бар және рұқсат беру лицензиясы бойынша қол жетімді артықшылығы болды. Нәтижесінде компаниялар оны өз жабдықтарына оңай бейімдеуге мүмкіндік алды. Сонымен қатар, бұл операциялық жүйеде негізінен Java бағдарламалау тілінде қосымшалар жасайтын көптеген әзірлеушілер қауымдастығы бар. Бірақ осының бәрімен, соңғы жылдар мұндай үстемдік ұзаққа созылмауы мүмкін екенін көрсетті және Android бәсекелестері оның нарықтағы үлесінің бір бөлігін қайтарып алуға тырысады.

1.6 Процестер

Барлық операциялық жүйелердегі негізгі ұғым - бұл процесс. Процесс, іс жүзінде, оны орындау кезінде бағдарлама болып табылады. Оның мекенжай кеңістігі әр процеспен байланысты - жад ұяшықтарының мекенжайларының нөлден белгілі бір максимумға дейінгі тізімі, процесс деректерді оқи алады және оны қайда жаза алады. Мекенжай кеңістігінде орындалатын бағдарлама, осы бағдарламаның деректері және оның стегі бар. Сонымен қатар, әр процеске регистрлер (соның ішінде, командалық санауыш және стек көрсеткіші), ашық файлдардың тізімі, өңделмеген ескертулер, байланысты процестердің тізімі және бағдарлама процесінде қажет барлық басқа ақпарат кіретін ресурстар жиынтығы кіреді. Сонымен, процесс - бұл бағдарламаның жұмысына қажетті барлық ақпаратты қамтитын контейнер.

Егер процесс осылай тоқтатылса, кейінірек ол тоқтатылған күйден қайта басталуы керек. Бұл тоқтата тұру кезеңінде процесс туралы барлық ақпарат бір жерде сақталуы керек дегенді білдіреді. Мысалы, процесс бір уақытта бірнеше файлды оқуға ашық болуы мүмкін. Осы файлдардың әрқайсысына ағымдағы позиция көрсеткіші қосылған (яғни байт немесе жазба нөмірі, оны келесіде оқу керек). Процесс тоқтатылған кезде, барлық осы көрсеткіштер сақталуы керек, сондықтан процесті қалпына келтіргеннен кейін орындалатын оқу қоңырауы қажетті деректерді оқуға әкеледі. Көптеген операциялық жүйелерде әр процесс туралы барлық ақпарат, өзінің мекен-жай кеңістігінің мазмұнын қоспағанда, операциялық жүйенің кестесінде сақталады, ол процестер кестесі деп аталады және құрылымдардың массиві

(немесе байланысты тізім) болып табылады, қазіргі кездегі процестердің әрқайсысына бір.

Осылайша, процесс (оның ішінде тоқтатылған) әдетте жад бейнесі деп аталатын өзінің мекенжай кеңістігінен және оның регистрлерінің мазмұны бар процестер кестесіне жазудан, сондай-ақ, процесті кейіннен қайта бастау үшін қажет басқа ақпараттан тұрады.

1.7 Мекенжай кеңістіктері

Әр компьютер орындалатын бағдарламаларды сақтау үшін қолданылатын белгілі бір жедел жадыны иемденеді. Қарапайым операциялық жүйелерде жадта тек бір бағдарлама бар. Екінші бағдарламаны іске қосу үшін алдымен біріншісін жою керек, содан кейін оның орнына екіншісін жадқа жүктеу керек.

Неғұрлым күрделі операциялық жүйелер бір уақытта бірнеше бағдарламаның жадында болуға мүмкіндік береді. Өзара кедергілерді (және операциялық жүйенің жұмысына кедергі) болдырмау үшін сізге қорғаныс механизмі қажет. Бұл механизм жабдықтың бөлігі болуы керек екеніне қарамастан, оны операциялық жүйе басқарады.

Жоғарыда аталған көзқарас компьютердің жедел жадын басқару және қорғау мәселелерімен байланысты. Басқа да, маңыздылығы кем емес жадыға байланысты мәселе бар - бұл процесстердің мекенжай кеңістігін басқару. Әдетте, әр процесс белгілі бір үздіксіз мекенжайлар жиынтығын, әдетте нөлден бастап максимумға дейін пайдалануға арналған. Қарапайым жағдайда процесске бөлінген мекенжай кеңістігінің максималды көлемі жедел жады көлемінен аз болады. Осылайша, процесс өзінің мекенжай кеңістігін толтыра алады және оны жедел жадқа орналастыру үшін жеткілікті орын болады.

Сонымен қатар, көптеген компьютерлерде 32 немесе 64 биттік адрестеу қолданылады, бұл сәйкесінше 232 немесе 264 байт мекенжай кеңістігін алуға мүмкіндік береді. Егер процестің мекен-жай кеңістігі компьютерде орнатылған жедел жадтан асып кетсе және процесс бүкіл кеңістікті толығымен пайдаланса не болады? Алғашқы компьютерлерде мұндай процесс әрдайым құлдырады. Қазіргі уақытта, жоғарыда айтылғандай, виртуалды жад технологиясы бар, онда операциялық жүйе мекенжай кеңістігінің бір бөлігін жедел жадта, ал бір бөлігін дискіде сақтайды, қажет болған жағдайда олардың бөліктерін ауыстырады. Негізінде, операциялық жүйе процесс сілтеме жасай алатын мекенжайлар жиынтығы ретінде мекенжай кеңістігінің абстракциясын жасайды. Мекенжай кеңістігі машинаның физикалық жадыдан бөлінген және одан үлкен немесе кішірек болуы мүмкін.

1.8 Файлдар

Барлық дерлік операциялық жүйелер қолдайтын тағы бір негізгі ұғым-файлдық жүйе. Жоғарыда айтылғандай, операциялық жүйенің негізгі функциясы-дискілер мен басқа енгізу-шығару құрылғыларының ерекшеліктерін жасыру және бағдарламашыға құрылғыларға тәуелсіз файлдардан тұратын ыңғайлы және түсінікті дерексіз модель ұсыну. Файлдарды құру, жою, оқу және жазу үшін жүйелік шақырулар қажет болатыны анық. Файл оқуға дайын болмай тұрып, оны дискіден тауып, ашып, оқығаннан кейін жабу керек. Бұл операцияларды жүргізу үшін жүйелік шақырулар қарастырылған.

Файлдарды сақтау орнын ұсыну үшін көптеген дербес компьютерлердің операциялық жүйелері каталогты файлдарды топтарға біріктіру әдісі ретінде пайдаланады.

Файл иерархиясы, процесс иерархиясы сияқты, ағаш түрінде ұйымдастырылған, бірақ ұқсастық осымен аяқталады. Процесс иерархиялары тереңдікте ерекшеленбейді (әдетте үш деңгейден аспайды), ал файл иерархиялары әдетте төрт, бес немесе одан да көп деңгейге ие. Процестердің иерархиялары қысқа мерзімге ие, көбінесе бірнеше минуттан аспайды, ал каталог иерархиясы бірнеше жылдар бойы өмір сүре алады. Процестер мен файлдарға қатысты анықтау және қорғау шаралары да әртүрлі. Әдетте, тек ата-ана процесі еншілес процесін басқара алады немесе тіпті оған қол жеткізе алады, бірақ файлдар мен каталогтарды тек иесіне ғана емес, сонымен қатар пайдаланушылардың кең тобына да оқуға мүмкіндік беретін механизмдер бар.

Каталог иерархиясына жататын әрбір файл иерархияның жоғарғы жағынан — түбірлік каталогтан бастап файлға баратын жолды көрсете отырып, оның толық атымен көрсетілуі мүмкін. Бұл абсолютті жол файлға жету үшін түбірлік каталогтан өту керек каталогтар тізімінен тұрады, мұнда қиғаш таңбалар (қиғаш сызық) құрамдас бөлгіштер ретінде қызмет етеді.

Кез-келген уақытта әр процестің ағымдағы жұмыс каталогы бар, оған қатысты қиғаш сызықтан басталмайтын файл жолдары қарастырылады. Процесс жаңа жұмыс каталогын анықтайтын жүйелік шақыруды қолдана отырып, жұмыс каталогын өзгерте алады.

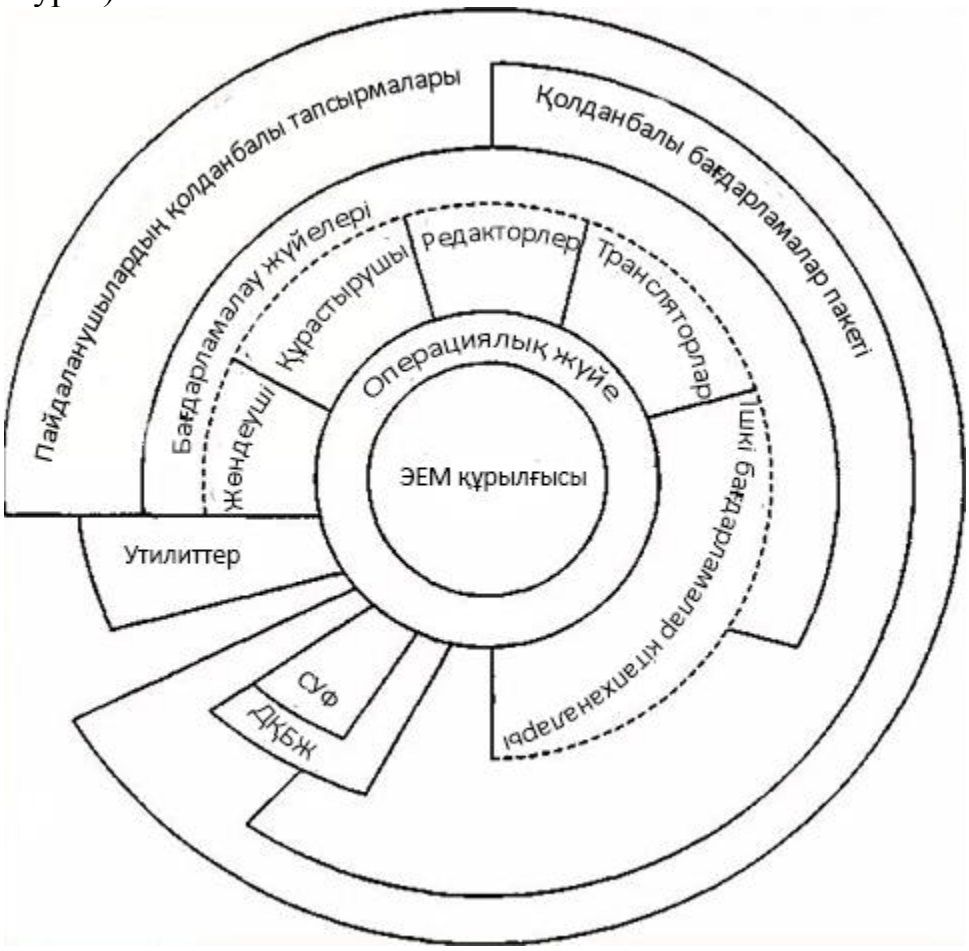
Жазу немесе оқу режимінде файлмен жұмыс жасамас бұрын, ол ашық болуы керек. Бұл кезеңде қол жеткізу құқықтары да тексеріледі. Егер кіруге рұқсат етілсе, жүйе келесі операцияларда қолданылатын файл дескрипторы деп аталатын бүтін санды қайтарады. Егер кіруге тыйым салынса, қате коды қайтарылады.

2 ОПЕРАЦИЯЛЫҚ ЖҮЙЕЛЕРДІҢ ЖІКТЕЛУІ

2.1 Бағдарламалық қамтамасыз ету түрлерін жіктеу

Ақпараттық технологиялардың қарқынды дамып келе жатқан нарығының қазіргі жағдайы мамандардан күрделі есептеу жүйелерінде болып жатқан процестерді түсінуді талап етеді. Жабдықтың жұмысын басқаратын бағдарламалық құрылымдардың құрылу және жұмыс істеу негіздерін білмей тиімді қосымшаларды құру және пайдалану мүмкін емес.

Жалпы жағдайда бағдарламалық жасақтаманы екі үлкен топқа бөлуге болады: компьютердің жұмысын басқаратын жүйелік бағдарламалар және пайдаланушының тапсырмаларын орындайтын қолданбалы бағдарламалар. Ең маңызды жүйелік бағдарлама - бұл операциялық жүйе, ол барлық жүйелік ресурстарды басқарады және қолданбалы бағдарламаларды жазуға негіз береді (4-сурет).



4 - сурет - Есептеу жүйесінің бағдарламалық қамтамасыз етуінің жалпыланған құрылымы

2.2 Операциялық жүйелердің мақсаты мен функциялары

Операциялық жүйе, ОЖ - бұл компьютерлік жабдықты пайдалануға мүмкіндік беретін бағдарламалар жиынтығы. ОЖ - бұл негізінен ресурстар әкімшісі, ол процессорларды, жадыны, енгізу-шығару құрылғыларын және деректерді басқарады.

ОЖ-нің негізгі мақсаты - ресурстарды басқару, ал ол басқаратын негізгі ресурстар-бұл компьютерлік жабдық:

- процессор;
- жады;
- енгізу-шығару құрылғылары.

ОЖ орындайтын негізгі функциялар:

- пайдаланушыдан тапсырмаларды немесе командаларды қабылдау және оларды өңдеу;
- бағдарламалық сұраныстарды қабылдау және орындау;
- орындауға жататын бағдарламаларды жедел жадыға жүктеу;
- бағдарламаны инициализациялау;
- барлық бағдарламалар мен деректерді сәйкестендіру;
- файлдарды басқару жүйелерінің жұмысын қамтамасыз ету;
- барлық енгізу/шығару операцияларын ұйымдастыру және басқару;
- жадыны бөлу және виртуалды жадыны ұйымдастыру;
- тапсырмаларды жоспарлау және диспетчерлеу;
- бағдарламалар арасында хабарламалар мен деректер алмасу механизмдерін ұйымдастыру;
- бір бағдарламаны екіншісінің әсерінен қорғау;
- жүйенің ішінара істен шығуы жағдайында қызметтер көрсету;
- бағдарламалау жүйелерінің жұмысын қамтамасыз ету.

Операциялық жүйелерге қойылатын талаптар ОЖ қандай функцияларды атқаратындығына байланысты, ол белгілі бір пайдалану талаптарын қанағаттандыруы керек, атап айтқанда, жүйе келесі қасиеттерге ие болуы керек:

а) Сенімділік. Жүйе ол жұмыс істейтін жабдық сияқты сенімді болуы керек. Бағдарламалық қамтамасыз етуде немесе аппараттық құралда қате пайда болған жағдайда, жүйе қатені анықтап, жағдайды түзетуге тырысуы керек немесе зиянды азайтуға тырысуы керек;

ә) Қорғау. Жүйе рұқсатсыз кіруден қорғалуы тиіс;

б) Тиімділік. ОЖ - бұл аппараттық ресурстардың едәуір бөлігін өз қажеттіліктері үшін пайдаланатын бағдарламалық жасақтаманың күрделі жиынтығы. Демек, ресурстардың көп бөлігі пайдаланушылардың иелігінде қалуы үшін жүйенің өзі мүмкіндігінше үнемді болуы керек. Сонымен қатар,

жүйе пайдаланушылардың ресурстарын жұмыс уақытын азайту үшін немесе ресурстың максималды жүктемесіне қол жеткізу үшін басқаруы керек;

в) Болжамдылық. Пайдаланушы ұзақ уақыт бойы техникалық қызмет көрсету тым көп өзгермегенін қалайды. Атап айтқанда, бағдарламаны іске қосу кезінде пайдаланушы алдыңғы тәжірибеге сүйене отырып, нәтижелерді қашан күтуге болатындығы туралы түсінікке ие болуы керек;

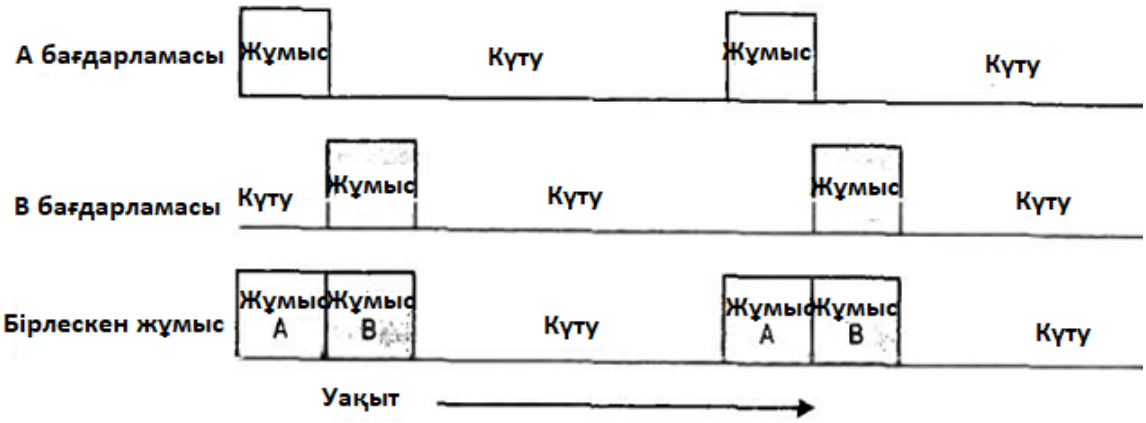
г) Ыңғайлылық. ОЖ жеткілікті икемді және ыңғайлы болуы керек.

2.3 Мультибағдарламалау. Уақытты бөлу режимі

Қазіргі ОЖ-де пайдаланушы тапсырмаларының өнімділігі мен жылдамдығын арттыру үшін мультибағдарламалау және уақытты бөлу режимдері қолданылады.

Мультибағдарламалау режимі - бір процессорда кезек-кезек орындалатын бірнеше бағдарлама компьютердің жадында орналасқан есептеу процесін ұйымдастыру әдісі. Сондай-ақ, мультибағдарламалауды есептеу жүйесінің жұмыс режимі ретінде анықтауға болады, онда бір процесс оған қажетті ресурстың босатылуын күтеді, ал басқа процесс бір уақытта процессордың ресурстарын алады. Бұл режимнің мәнін келесі мысалмен суреттеуге болады.

Сериялық процессорда жұмыс істейтін жүйеде екі дайын процесс бар: А процесі және В процесі (5-сурет).



5-сурет - Операциялық жүйенің көп тапсырмалы жұмыс режимінде А және В процестерінің орындалу диаграммасы

А процесі енгізу/шығару операциясын орындау қажет болған кезде В процесі CPU орталық процессорының ресурсын алады.

Уақытты бөлу режимі есептеу жүйесінің көп тапсырмалы жұмыс режимінің нұсқасы ретінде келесіге дейін азаяды:

- әр процестің өзіндік ресурстар кеңістігі бар;

- CPU іске қосылған процестермен қысқа командаларды алмастырады;
- Операциялық жүйе CPU мен процестер арасында командаларды беруді жүзеге асырады және осы алмасудың ретін басқарады.

2.4 Көп пайдаланушының жұмыс режимі. Нақты уақыт режимдері

Есептеу жүйелерінің көп пайдаланушы режимі, сондай-ақ мультибағдарламалау болып, әр процесте өзінің ресурстық кеңістігінің болуымен, операциялық жүйе деңгейінде хабар алмасу және басқару әдістерінің болуымен сипатталады. Қосымша мүмкіндіктер есептеу жүйесінің барлық ресурстарын тек бір пайдаланушының процестері арасында ғана емес, сонымен қатар жүйеде бір уақытта болатын әртүрлі пайдаланушылардың басқару сессиялары арасында бөлу арқылы беріледі.

Қазіргі заманғы операциялық жүйелерде процестердің кейбір түрлерінің сапасы мен жылдамдығын жақсарту үшін басқару нақты уақыт режимінде жүзеге асырылады. Есептеу жүйесін басқарудың осы әдісінің мәні асыруға болмайтын белгілі бір уақыт аралығында түсетін тапсырмаларды өңдеуді қамтамасыз етуде.

Нақты уақыт жүйелері үшін уақыт оларға негізгі параметр болып табылатындығымен сипатталады. Мысалы, өндірістік процестерді басқару жүйелерінде нақты уақыт режимінде жұмыс істейтін компьютерлер процесс туралы ақпарат жинап, оларды кәсіпорындағы машиналарды басқару үшін пайдалануы керек. Көбінесе олар өте қатаң уақытша талаптарға сай болуы керек. Мысалы, автомобиль құрастыру құбыры бойымен қозғалғанда, белгілі бір уақытта нақты операциялар жүргізілуі керек. Егер, мысалы, дәнекерлеу роботы дәнекерлеуді алдын ала немесе кеш бастаса, машина жарамсыз болады. Егер операцияны дәл уақытында (немесе белгілі бір уақыт аралығында) жүргізу керек болса, онда біз нақты уақыт жүйесімен айналысамыз. Көптеген ұқсас жүйелер өндірістік процестерді басқаруда, авиациялық-ғарыштық электронды жабдықта, әскери және басқа да ұқсас қосымшаларда кездеседі. Бұл жүйелер белгілі бір әрекеттер белгілі бір уақытта жүзеге асырылатынына абсолютті кепілдік беруі керек.

Мұндай жүйелердің тағы бір түрі - жұмсақ нақты уақыт жүйесі, ол қалаусыз болса да, түзетілмейтін, зиян келтірмейтін кез келген әрекеттің мерзімін сақтамауға өте қолайлы. Бұл санатқа сандық аудио немесе мультимедиалық жүйелер кіреді. Смартфондар сонымен қатар, нақты уақыттағы жұмсақ жүйелер болып табылады.

Нақты уақыттағы жүйелерге өте қатаң талаптар қойылатындықтан, кейде операциялық жүйелер қарапайым қолданбалы кітапхана болып табылады, онда барлығы бір-бірімен тығыз байланысты және жүйенің

бөліктері арасында ешқандай қорғаныс жоқ. Мұндай жүйенің мысалы eCos болуы мүмкін.

Қалта дербес компьютерлеріне, ендірілген жүйелерге және нақты уақыт жүйелеріне арналған операциялық жүйелердің санаттары өздеріне тән белгілері бойынша бір-бірімен айтарлықтай сәйкес келеді. Олардың барлығында, кем дегенде, нақты уақыттағы жұмсақ жүйелердің кейбір аспектілері бар. Ендірілген жүйелер мен нақты уақыттағы жүйелер тек осы жүйелерді жасаушылар салған бағдарламалық жасақтамамен жұмыс істейді; пайдаланушылар бұл арсеналға өздерінің бағдарламалық жасақтамаларын қоса алмайды, бұл қорғаныс мәселелерін шешуді айтарлықтай жеңілдетеді. Қалта дербес компьютерлері және ендірілген жүйелер жеке тұтынушыларға арналған, ал нақты уақыт жүйелері өнеркәсіптік өндірісте жиі қолданылады. Осыған қарамастан, олардың белгілі бір жалпы белгілері бар.

3 Процестерді ұйымдастыру және басқару

3.1 Процесс моделі

Бұл модельде компьютерде орындалатын барлық бағдарламалық қамтамасыз ету, кейде операциялық жүйені қоса, бірқатар дәйекті процестерге немесе қысқаша айтқанда, жай процестерге дейін азаяды. Процесс - бұл жай ғана орындалатын бағдарламаның мысалы, оның ішінде командалық санауыштың, регистрлердің және айнымалылардың ағымдағы мәндері. Тұжырымдамалық тұрғыдан алғанда, әр процестің өзіндік, виртуалды, орталық процессоры бар. Әрине, нақты орталық процессор процестер арасында үнемі ауысып отырады, бірақ жүйені түсіну үшін орталық процессордың бағдарламалар арасында қалай ауысатынын бақылауға тырысқаннан гөрі параллель режимде (жалған) жұмыс істейтін процестер жиынтығы туралы ойлау оңайырақ. Процестер арасындағы бұл тұрақты ауысу мультибағдарламалау немесе көп тапсырмалы жұмыс режимі деп аталады.

Орталық процессор процестер арасында қайта қосылып отыратындықтан, процестің есептеу жылдамдығы бірдей болмайды және егер сол процесс қайтадан іске қосылса, оны қайта көрсету мүмкін болмайды. Сондықтан процестер олардың орындалу уақытына қатысты қатаң белгіленген болжамдарды қолдана отырып бағдарламаланбауы керек. Мысалы, басқа құрылғыда жұмыс істейтін жоғары сапалы бейнені сүйемелдеу үшін музыка ойнайтын аудио процесті қарастырайық. Дыбысты бейнеден сәл кешірек іске қосуға болатындықтан, аудио процесс бейне серверге ойнатуды бастау туралы сигнал береді, содан кейін дыбысты ойнатпас бұрын бос циклды 10 000 рет бастайды. Егер цикл сенімді таймер ретінде қызмет етсе, онда бәрі дұрыс болады, бірақ егер процессор бос циклды орындау кезінде басқа процеске ауысуды шешсе, тиісті кадрлар көрсетілген кезде аудио процесс жалғасуы мүмкін, өкінішке орай, бейне мен аудионы синхрондау төмендейді. Процестің нақты уақыт ауқымына қатысты оның жұмысы үшін осындай маңызды талаптар болған кезде, белгілі бір миллисекундтан кейін нақты оқиғалар орын алуы керек және олардың пайда болуы үшін арнайы шаралар қабылдануы керек. Бірақ, әдетте, көптеген процестерге орталық процессордың белгіленген Көп пайдаланушы режимі де, әртүрлі процестердің салыстырмалы жылдамдығы да әсер етпейді.

3.2 Процесті құру

Техникалық тұрғыдан алғанда, барлық жағдайларда жаңа процесс процесті құруға арналған жүйелік шақыруды орындайтын бұрыннан бар процесс арқылы жасалады. Бұл процесс жұмыс істейтін пайдаланушы процесі, пернетақта немесе тінтуір оқиғаларынан туындаған жүйелік процесс немесе пакеттік тапсырмаларды басқару процесі болуы мүмкін. Бұл процесс жаңа процесті құру үшін жүйелік шақыруды жүзеге асырады. Бұл жүйелік шақыру операциялық жүйеге жаңа процесті құруды бұйырады және тікелей немесе жанама түрде қай бағдарламаны іске қосуды көрсетеді.

UNIX - те жаңа процесті құру үшін бір ғана жүйелік қоңырау бар, бұл - fork. Бұл шақыру шақыру процесінің нақты көшірмесін жасайды. Fork жүйелік шақыруын орындағаннан кейін екі процесс, яғни ата-ана және енші, бірыңғай жады кескініне, конфигурация сипаттамасының бірыңғай жолдарына және бірдей ашық файлдарға ие. Бұдан басқа ештеңе. Әдетте, осыдан кейін енші процесі жады кескінін өзгертеді және execve немесе оған ұқсас жүйелік шақыруды орындау арқылы жаңа бағдарламаны бастайды. Мысалы, пайдаланушы қабықта sort пәрменін терген кезде, қабық sort пәрмені орындалатын бұтақты енші процесін жасайды. Бұл екі сатылы процестің мәні - енші процесіне тармақталғаннан кейін оның файл дескрипторларын басқаруға мүмкіндік беру, бірақ стандартты кіріс, стандартты шығыс және қате туралы хабарламалардың стандартты шығуын қайта бағыттау мақсатында execve жасамас бұрын.

Windows-та бәрі басқаша болады: Win32 createprocess функциясының бір шақыруы процесті жасайды және оған қажетті бағдарлама жүктеледі. Бұл шақыруда 10 параметр бар, оның ішінде орындалатын бағдарлама, осы бағдарлама үшін пәрмен жолының параметрлері, әртүрлі қауіпсіздік параметрлері, ашық файлдардың мұрагерлігін басқаратын биттер, басымдықтар туралы ақпарат, процесс үшін жасалған терезе сипаттамасы (егер ол қолданылса) және шақырушы бағдарлама жаңа құрылған процесс туралы ақпаратты қайтаратын құрылымға сілтегіш бар. Win32-де CreateProcess функциясынан басқа, процестерді басқаруға және оларды синхрондауға, сондай-ақ, онымен байланысты барлық нәрсені орындауға арналған 100-ге жуық басқа функциялар бар.

Екі жүйеде де, UNIX және Windows, процесті құрғаннан кейін, ата-ана мен енші процестерінің өзіндік, жеке мекенжай кеңістігі болады. Егер қандай да бір процесс өзінің мекенжай кеңістігінде сөзді өзгертсе, басқа процестерге бұл өзгерістер көрінбейді. UNIX-те енші процесінің мекенжай кеңістігінің бастапқы күйі - бұл ата-ана процесінің мекенжай кеңістігінің көшірмесі, бірақ бұл мүлдем басқа мекенжай кеңістігі - олардың деректерді жазуға ортақ

жадысы жоқ. UNIX-тің кейбір енгізулері бағдарламаның мәтінін оны өзгерту мүмкіндігінсіз процестер арасында бөледі. Сонымен қатар, бала процесі ата-ана процесінің барлық жадысын бөлісе алады, бірақ егер жады жазу кезінде көшіру режимінде (жазу кезінде көшіру) бөлісілсе, бұл кез-келген процестің жадының бір бөлігін өзгертуге тырысқан сайын, бұл бөлік тек жабық жады аймағында модификацияға кепілдік беру үшін алдымен нақты көшіріледі. Сондай-ақ, жазу режимінде қолданылатын жады ортақ пайдалануға жатпайтынын атап өткен жөн.

Алайда, жаңадан құрылған процесс өз жасаушысымен ашық файлдар сияқты басқа ресурстардың бір бөлігін жасай алады. Windows жүйесінде ата-ана мен еншілес процестерінің мекенжай кеңістігі басынан бастап ерекшеленеді.

3.3 Процесті аяқтау

Құрылғаннан кейін процесс жұмыс істей бастайды және өз міндетін орындайды. Бірақ ештеңе мәңгілікке созылмайды, тіпті процестер. Ерте ме, кеш пе, жаңа процестер әдетте келесі жағдайларға байланысты аяқталады:

- әдеттегі шығу (ерікті);
- қате пайда болған кезде шығу (ерікті);
- фатальді қатенің пайда болуы (еріксіз);
- басқа процеспен жойылу (еріксіз).

Көптеген процестер жұмыс аяқталғаннан кейін аяқталады. Компилятор өзіне берілген бағдарламаны құрастырған кезде, ол операциялық жүйеге өзінің жұмысының аяқталғаны туралы хабарлайтын жүйелік шақыруды орындайды. UNIX-тегі бұл шақыру - exit, ал Windows-та - ExitProcess. Экранмен жұмыс істейтін бағдарламалар да өз еркімен аяқтауды қолдайды. Мәтіндік процессорлар, интернет-шолғыштар және ұқсас бағдарламалар әрқашан пайдаланушы ашылған барлық уақытша файлдарды жоюды және олардың жұмысын аяқтауды бұйыруға болатын белгішені немесе мәзір элементін қамтиды.

Аяқталудың екінші себебі - фатальді қатеге әкелетін қатені анықтау. Мысалы, егер пайдаланушы

foo.c бағдарламасын құрастыру мақсатында cc foo.c пәрменін терсе, ал мұндай атаумен файл болмайды, онда осы фактінің болуы және компилятордан шығу туралы қарапайым хабарландыру болады. Қате параметрлерді орнату кезінде экранды пайдаланатын интерактивті процестерден шығу әдетте болмайды. Оның орнына параметрлерді қайта енгізуді сұрайтын сұхбат терезесі пайда болады.

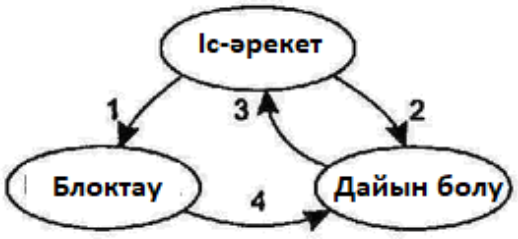
Аяқтаудың үшінші себебі - процестің өзінен туындаған қате, көбінесе бағдарламадағы қатеге байланысты. Мысал ретінде дұрыс емес нұсқауларды, жоқ жады мекенжайына сілтемені немесе нөлге бөлуді келтіруге болады. Кейбір жүйелерде (мысалы, UNIX) процесс операциялық жүйеге белгілі бір қателерді өз бетінше өңдеу ниеті туралы хабарлай алады, бұл жағдайда осындай қателіктердің бірі пайда болған кезде процесс сигнал алады (үзіледі) және аяқталмайды.

Процестің аяқталуының төртінші себебі - жүйелік шақыру процесін орындау, ол операциялық жүйеге басқа процестерді аяқтауға бұйрық береді. UNIX-те бұл қоңырау kill деп аталады. Тиісті Win32 функциясы TerminateProcess деп аталады. Екі жағдайда да аяқтауға себеп болатын процестің тиісті өкілеттіктері болуы керек. Кейбір жүйелерде процестің ерікті немесе еріксіз аяқталуымен оның құрған барлық процестері дереу аяқталады. Бірақ LINUX пен Windows-та бұндай жоқ.

3.4 Процесс күйлері

6-суретте процесті орналастыруға болатын үш күйді көрсететін диаграмма көрсетілген:

- орындалатын (қазіргі уақытта орталық процессорды пайдалану);
- дайын (жұмыс істейді, бірақ басқа процесті орындауға мүмкіндік беру үшін уақытша тоқтатылған);
- блокталған (сыртқы оқиға пайда болғанша орындалмайды).



1. Кіріс күту кезінде процесс блокталған
2. Диспетчер басқа процесті таңдады
3. Диспетчер бұл процесті таңдады
4. Кіріс қолжетімді болды

6-сурет - Процесс орындалатын, блокталған немесе дайын күйде болуы мүмкін. Көрсеткілер осы күйлер арасындағы өтулерді көрсетеді

Логикалық тұрғыдан алғанда, алғашқы екі күй бір-біріне ұқсас. Екі жағдайда да процесс орындалғысы келеді, бірақ екінші жағдайда бұл үшін уақытша қолжетімді процессор жоқ. Үшінші күй алғашқы екеуінен түбегейлі ерекшеленеді, өйткені процессор бұдан басқа ештеңе жасамаса да, процесті орындау мүмкін емес.

6-суретте көрсетілгендей, осы үш күй арасында төрт ауысу болуы мүмкін. 1-ауысу операциялық жүйе процестің қазіргі уақытта орындалмайтындығын анықтаған жағдайда орын алады. Кейбір жүйелерде блокталған күйге өту үшін процесс pause сияқты жүйелік шақыруды орындай алады. Басқа жүйелерде, соның ішінде UNIX-те, процесс арнадан немесе арнайы файлдан (мысалы, терминалдан) оқылған кезде және қолжетімді кірістер болмаса, процесс автоматты түрде блокталады.

2 және 3-өтулерді процестің өзінің ескертуінсіз процесті жоспарлаушы шақырады, ол операциялық жүйенің бөлігі болып табылады. 2-ші ауысу жоспарлаушы орындалатын процесс жеткілікті түрде алға жылжыды деп шешкен кезде пайда болады және басқа процессорға орталық процессордың жұмыс уақытының үлесін алуға мүмкіндік беретін уақыт келді. 3-ші ауысу барлық басқа процестер уақыттың үлесін алған кезде пайда болады және орталық процессорды оның орындалуын қайта бастау үшін бірінші процеске беру сәті келді. Жоспарлау мәселесі, яғни шешім, қандай процесс, қашан және қанша уақыт орындалуы керек, өте маңызды рөл атқарады және сәл кейінірек осы тарауда қарастырылады. Тұтастай алғанда жүйенің тиімділігіне және жеке процеске әділ көзқарасқа сәйкес келетін бәсекелес талаптарды теңдестіру үшін көптеген алгоритмдер ойлап табылды.

4-ауысу, егер процесс күткен сыртқы оқиға орын алса (мысалы, кіріс деректерінің түсуі) жүзеге асырылады. Егер осы уақытқа дейін орындалатын басқа процестер болмаса, 3-ші ауысу шақырылып, процесс қайта басталады. Әйтпесе, ол орталық процессор қолжетімді болғанша және оның кезегі келгенше дайын күйде біраз күтуге тура келеді.

3.5 Ағындарды қолдану

Ағындар деп аталатын осындай шағын процестерге қажеттілік бірқатар себептерге байланысты. Олардың кейбірін қарастырайық. Ағындарды қолданудың негізгі себебі - көптеген қосымшаларда бір уақытта бірнеше әрекеттер болады, олардың бір бөлігі мерзімді түрде блокталуы мүмкін. Бағдарламалау моделі мұндай қосымшаны квази-параллель режимде орындалатын бірнеше тізбектелген ағындарға бөлу арқылы жеңілдетілген.

Біз осындай дәлелдермен осыған дейін кезіккен едік. Олар процестерді құруды қолдау үшін қолданылған. Үзілістер, таймер және контекстік қосқыштар туралы ойланудың орнына, параллель процестер туралы ойлауға болады. Бірақ қазір ғана ағындарды қарастыра отырып, біз жаңа элементті қосамыз: параллель процестерді бірыңғай мекенжай кеңістігі мен барлық қол жетімді деректерді пайдалану мүмкіндігі. Бұл мүмкіндік бірнеше процестерді

қолдануға жарамсыз қосымшалар үшін өте маңызды рөл атқарады (олардың жеке мекенжай кеңістігі бар).

Ағындардың пайдасына екінші дәлел - «ауыр» процестермен салыстырғанда оларды құру мен жоюдың жеңілдігі (яғни жылдамдығы). Көптеген жүйелерде ағындарды құру процестерді құруға қарағанда 10-100 есе жылдам. Бұл қасиет әсіресе ағындар санын тез және қарқынды өзгерту қажет болған кезде пайдалы.

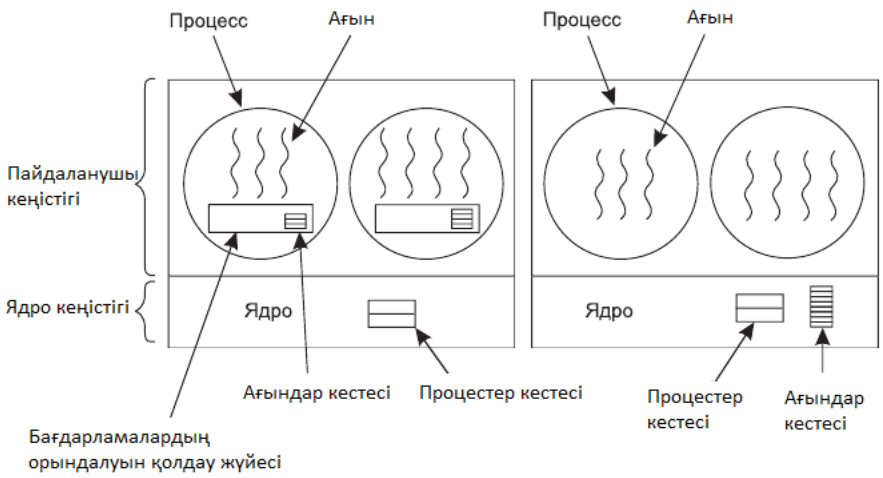
Ағындардың пайдасына үшінші дәлел өнімділікке де қатысты. Ағындар бір орталық процессордың бөлігі ретінде жұмыс істеген кезде, олар ешқандай өнімділікті арттырмайды, бірақ айтарлықтай есептеулер жүргізіліп, уақыттың көп бөлігі кіріс-шығыс күтуге жұмсалған кезде, ағындардың болуы бұл әрекеттерді қосымшаның жұмысын тездететін уақытқа сәйкес келтіруге мүмкіндік береді. Сонымен, ағындар параллель есептеудің нақты мүмкіндігі бар бірнеше орталық процессорлары бар жүйелер үшін өте пайдалы.

3.6 Пайдаланушы кеңістігінде ағындарды іске асыру

Ағындар жиынтығын жүзеге асырудың екі негізгі орны бар: пайдаланушы кеңістігінде және ядрода. Бұл мәлімдеме біршама даулы, өйткені гибридті іске асыру да мүмкін. Енді біз бұл әдістерді олардың барлық артықшылықтары мен кемшіліктерімен сипаттаймыз.

Бірінші әдіс - ағындардың барлық жиынтығын пайдаланушы кеңістігіне орналастыру. Ядро бұл жиынтық туралы ештеңе білмейді. Ядроға келетін болсақ, ол қарапайым, бір ағынды процестерді басқарады. Бірінші және айқын артықшылығы - пайдаланушы деңгейіндегі ағындар жиынтығын ағындарды қолдамайтын операциялық жүйеде іске асыруға болады. Барлық операциялық жүйелер, тіпті әлі де дамып келе жатқан жүйелер де осы санатқа жатады. Бұл тәсілде ағындар кітапхананың көмегімен жүзеге асырылады.

Барлық осы енгізулер бірдей жалпы құрылымға ие (7,а -сурет). Ағындар бағдарламаларды орындауды қолдау жүйесінің (run-time system) жоғарғы жағында іске қосылады, бұл ағындарды басқаратын процедуралар жиынтығы. Олардың төртеуін: pthread_create, pthread_exit, pthread_join және pthread_yield - біз қарастырдық, бірақ әдетте жиынтықта басқа процедуралар бар.



а ә

7-сурет - Ағындар жиынтығы: а - пайдаланушы деңгейінде;
ә -ядромен басқарылатын

Ағындар пайдаланушы кеңістігінде басқарылған кезде, осы процестегі ағындарды бақылау үшін әр процестің өзіндік ағын кестесі болуы керек. Бұл кесте ядродағы процестер кестесінің аналогы болып табылады, оның құрамында тек әр ағынға тиесілі қасиеттер бар, мысалы, ағынды командалық есептегіш, стек көрсеткіші, регистрлер, күй және т.б. ағындар кестесі бағдарламаның орындалуын қолдау жүйесімен басқарылады. Ағын дайын күйге ауысқанда немесе блокталғанда, оның орындалуын қалпына келтіру үшін қажетті ақпарат ағындар кестесінде сақталады, өйткені ядро процестер кестесінде процестер туралы ақпаратты сақтайды.

Ағын оның жергілікті блокталуын тудыруы мүмкін әрекеттерді жасаған кезде, мысалы, оның процесінің басқа ағымы қандай да бір жұмысты аяқтағанша күту, ол бағдарламаның орындалуын қолдау жүйесінің процедурасын тудырады. Бұл процедура ағынды блоктау күйіне ауыстыруға болатындығын тексереді. Мүмкін болса, ол ағындар кестесінде ағын регистрлерін (яғни меншікті регистрлерді) сақтайды, кестеде орындауға дайын ағынды табады және машина регистрлерін жаңа ағынның сақталған мәндерімен қайта жүктейді. Стек көрсеткіші мен командалық санауыш ауыстырылғаннан кейін жаңа ағынның орындалуы автоматты түрде жалғасады. Егер машинада барлық регистрлерді сақтау туралы нұсқаулық берілсе және келесі нұсқаулық барлық регистрлерді жүктеу болса, онда ағынды толығымен ауыстыру бірнеше нұсқауларға байланысты жүзеге асырылуы мүмкін. Осылайша жүзеге асырылған ағындарды ауыстыру, кем дегенде, тапсырыс бойынша немесе одан да көп, негізгі басқаруды ұстап

қалудан тезірек, бұл пайдаланушы деңгейінде жүзеге асырылатын ағындар жиынтығының пайдасына күшті дәлел болып табылады.

Бірақ ағындардың процестерден бір негізгі айырмашылығы бар. Ағын оның орындалуын уақытша тоқтатқан кезде, мысалы, thread_yield шақырған кезде, thread_yield процедуралық коды ағын туралы ақпаратты ағындар кестесінде өздігінен сақтай алады. Сонымен қатар, содан кейін ол басқа ағынды таңдау үшін ағынды жоспарлаушыны шақыра алады. Ағынның күйін сақтайтын процедура және жоспарлаушы-бұл жергілікті процедуралар, сондықтан оларды шақыру ядроға қарағанда әлдеқайда тиімді. Сонымен қатар, trap нұсқаулығымен жүзеге асырылатын негізгі бақылауды ұстап қалудың қажеті жоқ, контекстті ауыстыру қажет емес, жадыдағы кэшті дискіге тастаудың қажеті жоқ және т.б. осының арқасында ағынды жоспарлаушы өте тез жұмыс істейді.

Пайдаланушы деңгейінде жүзеге асырылатын ағындардың басқа да артықшылықтары бар. Олар әр процесті жоспарлау алгоритмінің жеке параметрлеріне ие болуға мүмкіндік береді. Мысалы, қоқыс жинағыш ағыны бар кейбір қосымшалар үшін тағы бір плюс бар - олар дұрыс емес уақытта тоқтатылған ағындар туралы алаңдамауы керек. Бұл ағындар жақсырақ масштабталады, өйткені ядро жадысындағы ағындар ядрода кесте мен стек үшін кеңістікті қажет етеді, бұл өте көп ағынмен қиындық тудыруы мүмкін. Бірақ жақсы өнімділікке қарамастан, пайдаланушы деңгейінде жүзеге асырылатын ағындар бірқатар маңызды проблемаларға ие. Олардың біріншісі - блоктайтын жүйелік шақыруларды қалай жүзеге асыру. Кейбір пернелерді баспас бұрын ағын ақпаратты пернетақтадан оқиды деп елестетіп көріңіз. Біз ағынға нақты жүйелік шақыруды орындауға рұқсат бере алмаймыз, өйткені бұл барлық ағындардың орындалуын тоқтатады. Ағындарды ұйымдастырудың негізгі мақсаттарының бірі, ең алдымен, әр ағынға блокталған шақыруларды қолдануға мүмкіндік беру болды, бірақ бір блокталған ағынның басқа ағындардың орындалуына әсерін болдырмады. Блоктайтын жүйелік шақырулармен жұмыс жасай отырып, бұл мақсатқа еш қиындықсыз қалай жетуге болатындығын түсіну өте қиын.

Барлық жүйелік шақыруларды өзгертуге және блокталмайтын шақыруларға айналдыруға болады (мысалы, пернетақтадан оқу, егер буферде таңбалар болмаса, байт нөлін қайтарады), бірақ бұл үшін операциялық жүйеге енгізілетін өзгерістер ынта тудырмайды. Сонымен қатар, пайдаланушы деңгейінде жүзеге асырылатын ағындарды пайдалану үшін дәлелдердің бірі - оларды қолданыстағы операциялық жүйелердің басқаруымен орындауға болатындығы. Сонымен қатар, read жүйелік шақыру семантикасын өзгерту көптеген пайдаланушы бағдарламаларын өзгертуді қажет етеді.

Егер шақырудың блокталатынын алдын ала хабарлау мүмкіндігі болса, басқа балама бар. UNIX-тің көптеген нұсқаларында select жүйелік шақыруы бар, ол шақыруға жоспарланған read жүйелік шақыруы блокталатындығын айтуға мүмкіндік береді. Егер мұндай шақыру болса, read кітапханалық процедурасын жаңа процедурамен алмастыруға болады, ол алдымен select процедурасын шақырады, содан кейін ғана қауіпсіз болса, read шақырады (яғни блоктауды орындамайды). Егер read шақыруы блокталса, ол орындалмайды. Оның орнына басқа ағынның орындалуы басталады. Келесі жолы, бағдарламаны орындауды қолдау жүйесі басқаруды алған кезде, бұл жолы read шақыруы қауіпсіз болатындығын қайта тексере алады. Бұл тәсілді жүзеге асыру үшін жүйелік шақырулар кітапханасының кейбір бөліктерін қайта жазу қажет, оны тиімді және талғампаз шешім ретінде қарастыруға болмайды, бірақ бұл да нұсқалардың бірі. Тексеру мақсатында жүйелік шақырудың айналасында орналасқан код конверт (jacket) немесе қабық немесе орауыш (wrapper) деп аталады.

Жүйелік шақыруларды блоктау мәселесі парақтың болмау қатесін біршама қайталайды. Егер бағдарлама жадыда жоқ нұсқауларды шақырса (немесе нұсқауларға ауысса), жетіспейтін бетке жүгіну қатесі пайда болады және операциялық жүйе дискіге кіріп, жетіспейтін нұсқауларды (және олардың көршілерін) алады. Бұл жетіспейтін бетті шақыру қатесі деп аталады. Процесс қажетті нұсқаулық табылып, оқылғанша блокталады. Егер ағынды орындау кезінде жетіспейтін бетке жүгіну қатесі пайда болса, онда сіз күткендей ағындардың бар екендігі туралы білмейтін ядро дискіні енгізу-шығару операциясы аяқталғанға дейін бүкіл процесті блоктайды, тіпті басқа ағындар орындауға дайын болса да.

Пайдаланушы деңгейінде жүзеге асырылатын ағындар жиынтығын пайдалану тағы бір проблемамен байланысты: егер ағындардың біреуін орындау басталса, онда бірінші ағын орталық процессорға өз еркімен берілгенге дейін осы процеске тиесілі басқа ағын орындалмайды. Бірыңғай процесс аясында процестердің жұмысын дөңгелек цикл бойынша жоспарлауға мүмкіндік беретін таймерде үзілістер жоқ (кезекпен). Егер ағын бағдарламаларды өз еркімен орындауды қолдау жүйесіне кірмесе, жоспарлаушының жұмыс істеуге мүмкіндігі болмайды.

Ағындардың шексіз орындалуы мәселесін басқаруды секундына бір рет таймерге сигнал (үзіліс) сұрау арқылы бағдарламаның орындалуын қолдау жүйесіне беру арқылы шешуге болады, бірақ бағдарлама үшін бұл ең жақсы шешім емес. Таймерді мерзімді түрде және жиі үзу мүмкіндігі әрдайым берілмейді, бірақ егер ол берілсе де, жалпы шығындар өте маңызды болуы мүмкін. Сонымен қатар, ағынға таймердің үзілуі қажет болуы мүмкін, бұл

таймерді бағдарламаның орындалуын қолдау жүйесіне кедергі келтіруі мүмкін.

Пайдаланушы деңгейінде жүзеге асырылатын ағындарға қарсы тағы бір күшті дәлел - бұл бағдарламашылар көбінесе блокталатын қосымшаларда, мысалы, көп ағынды веб-серверде қажет болады. Бұл ағындар жиі жүйелік шақырулар жасайды. Жүйелік шақыруды орындау үшін ядро басқаруды ұстап қалуды жүзеге асырғаннан кейін, егер алдыңғы ағын блокталған болса, ағындарды ауыстыру қиын болмайды, ал ядро осы мәселені шешумен айналысса, read жүйелік шақыруының қауіпсіздігін анықтау үшін select жүйелік шақыруына үнемі жүгінудің қажеті болмайды. Неліктен орталық процессордың жылдамдығына толығымен байланған және блоктауды сирек қолданатын қосымшаларда ағындарды пайдалану керек? Алғашқы N жай сандарды есептеу кезінде немесе шахмат ойнау кезінде ағындарды қолдануды ешкім байыпты түрде ұсынбайды, өйткені бұл жағдайда олардың пайдасы аз болады.

3.7 Ядродағы ағындарды іске асыру

Енді ядро ағындар туралы біліп, оларды басқарса не болатынын қарастырайық. 7, ә-суретте көрсетілгендей, мұнда бағдарламалардың орындалуын қолдау жүйесі қажет емес. Сондай-ақ, әр ағында процестер кестесі жоқ. Оның орнына ядрода жүйеде бар барлық ағындарды бақылайтын ағындар кестесі бар. Ағын жаңа ағынды жасау немесе қолданыстағы ағынды жою қажет болған кезде, ол ядродағы ағындар кестесін жаңарту арқылы құратын немесе бұзатын ядроға жүгінеді.

Ядродағы ағындар кестесінде әр ағынның регистрлері, күйі және басқа ақпарат бар. Барлық ақпарат пайдаланушы деңгейінде жасалған ағындар үшін пайдаланылғанға ұқсас, бірақ қазір ол пайдаланушы кеңістігінде емес, ядрода болады (бағдарламаның орындалуын қолдау жүйесінің ішінде). Бұл ақпарат дәстүрлі ядролардың бір ағынды процестерге қатысты қолдайтын ақпараттың ішкі жиынтығы, яғни процесс күйінің ішкі жиынтығы. Сонымен қатар, ядро оларды бақылау үшін дәстүрлі процестер кестесін қолдайды.

Ағынды блоктауға қабілетті барлық шақырулар жүйелік ретінде жүзеге асырылады, бағдарламаларды орындауды қолдау жүйесіндегі процедурадан гөрі айтарлықтай шығындар бар. Ағын блокталған кезде, өз қалауы бойынша ядро сол процестен басқа ағынды (егер дайын ағын болса) немесе басқа процестен ағынды бастай алады. Ағындар пайдаланушы деңгейінде іске асырылған кезде, бағдарламаны орындауды қолдау жүйесі ядро одан орталық процессорды алғанға дейін (немесе орындауға дайын бірде-бір ағын қалмайынша) өз процесінің іске қосылған ағындарымен жұмыс істейді.

Ядродағы ағындарды құру және жою салыстырмалы түрде едәуір шығындарды қажет ететіндіктен, кейбір жүйелер қалыптасқан жағдайды ескере отырып, дұрыс тәсіл қолданады және өз ағындарын қайта қолданады. Ағынды жою кезінде ол орындалмайды деп белгіленеді, бірақ бұл оның ядродағы деректер құрылымына әсер етпейді. Біраз уақыттан кейін, жаңа ағын пайда болған кезде, ескі ағын қайта іске қосылады, бұл уақытты үнемдеуге әкеледі. Ағындарды қайта пайдалануға пайдаланушы деңгейінде де рұқсат етіледі, бірақ бұл үшін жеткілікті негіз жоқ, өйткені ағындарды басқару шығындары әлдеқайда аз.

Ядро деңгейінде жүзеге асырылатын ағындар үшін жаңа, блоктайтын жүйелік шақырулар қажет емес. Сонымен қатар, егер орындалатын ағындардың бірі жетіспейтін бетке жүгіну қатесіне тап болса, ядро процестің кез-келген басқа дайын ағындарының болуын оңай тексере алады және егер бар болса, сұралған бетті дискіден шығаруды күту жалғасқанша олардың біреуін іске қосады. Бұл ағындардың басты кемшілігі жүйелік шақыруға айтарлықтай уақыт жұмсау болып табылады, сондықтан егер ағындар бойынша операциялар (құру, жою және т. б.) болса, жиі орындалады, бұл айтарлықтай шығындарға әкеледі.

Ядро деңгейінде құрылған ағындар бірқатар мәселелерді шешуге мүмкіндік бергенімен, олар барлық проблемаларды шеше алмайды. Мысалы, көп ағынды процесс тармақталған кезде не болады? Жаңа процестегі ағындар саны ескі процестегідей бола ма, әлде тек бір ғана ағын бола ма? Көптеген жағдайларда ең жақсы таңдау келесі процестің орындалуына байланысты болады. Егер ол жаңа бағдарламаны іске қосу үшін ехес командасын шақырғысы келсе, онда бір ғана ағынның болуы дұрыс таңдау болуы мүмкін. Бірақ егер ол орындалуды жалғастырса, онда барлық қолжетімді ағындарды көбейту жақсы болар еді.

Тағы бір мәселе - сигналдар. Сигналдар ағындарға емес, процестерге жіберілетінін есте ұстаған жөн, кем дегенде бұл классикалық модельде жасалады. Кіріс сигналы қандай ағындарды өңдеуі керек? Мүмкін, ағындар өздерінің мүдделерін нақты сигналдарға тіркеуі керек, сондықтан сигнал келіп түскен кезде ол осы сигналға өзінің қызығушылығын білдіретін ағынға берілуі керек пе? Сонда сұрақ туындайды: егер бір сигналға екі немесе одан да көп ағын тіркелген болса, не болады? Бұл тек ағындар тудыратын екі мәселе, бірақ іс жүзінде олар әлдеқайда көп.

4 ЕНГІЗУ-ШЫҒАРУДЫ БАСҚАРУ

4.1 Енгізу-шығару құрылғылары

Енгізу-шығару құрылғыларын екі санатқа бөлуге болады: блок құрылғылары және символдық құрылғылар. Блоктарға ақпаратты белгіленген ұзындықтағы блоктарда сақтайтын құрылғылар кіреді, олардың әрқайсысының өз мекенжайы бар. Әдетте блоктардың өлшемдері 512-ден 65,536 байтқа дейін өзгереді. Барлық деректерді беру бір немесе бірнеше бүтін (бірізді) блоктардан тұратын пакеттермен жүргізіледі. Блок құрылғысының маңызды қасиеті - ол барлық басқа блоктарға қарамастан әр блокты оқи немесе жаза алады. Ең көп таралған блок құрылғыларының қатарына қатты дискілер, Blu-ray дискілері және USB флэш-дискілері жатады.

Басқа түрін енгізу-шығару құрылғылары - символдық құрылғы. Олар блок құрылымына жатпайтын таңбалар ағынын шығарады немесе қабылдайды. Олар мақсатты емес және ешқандай позициялау әрекеті жоқ. Символдық құрылғылар ретінде принтерлер, желілік интерфейстер, тышқандар (көрсеткіш құрылғы ретінде) және диск құрылғыларына ұқсамайтын көптеген басқа құрылғылар қарастырылуы мүмкін. Бұл жіктеу схемасы жетілдірілмеген. Кейбір құрылғылар оған жатпайды. Мысалы, сағат блокқа бағытталған емес. Олар сонымен қатар символдық жолдарды жасамайды немесе қабылдамайды. Олардың барлығы белгілі бір уақыт аралығында үзілістерді тудырады. Жадыда дисплейі бар экрандар да осы модельге сәйкес келмейді. Дәл сол себепті сенсорлық экрандар оған жатпайды. Дегенмен, блоктық және символдық құрылғылардың моделі оны операциялық жүйенің бағдарламалық жасақтамасының бір бөлігін енгізу-шығару құрылғысынан тәуелсіз ету үшін негіз ретінде пайдалану үшін жеткілікті.

4.2 Құрылғы контроллері

Енгізу-шығару құрылғылары көбінесе механикалық және электронды компоненттерден тұрады. Көбінесе модульдік дизайнды алу және құрылғыға жалпы көрініс беру үшін осы екі компонентті бөлуге болады. Электрондық компонент құрылғы контроллері немесе адаптер деп аталады. Жеке компьютерлерде ол көбінесе кеңейту ұясына (PCIe) салынған жүйелік тақтадағы микросхема немесе баспалық плата түрінде болады. Механикалық компонент құрылғының өзімен ұсынылған.

Контроллердің міндеті - сериялық бит ағынын байт блогына түрлендіру және қажет болған жағдайда қателерді түзету. Байт блогы әдетте

контроллердің құрамына кіретін буферде бастапқы биттік құрастырудан өтеді. Блоктың тексеру сомасын тексеріп, оны қатесіз деп жариялағаннан кейін оны жедел жадыға көшіруге болады.

Сұйық кристалды дисплейге негізделген монитор контроллері сонымен қатар төмен деңгейде биттік сериялық құрылғы ретінде жұмыс істейді. Ол жадыдан көрсетілуі керек таңбалары бар байттарды оқиды және оларды экранға жазу үшін тиісті пиксельдердің артқы жарығының поляризациясын өзгерту үшін қолданылатын сигналдарды жасайды. Егер дисплей контроллері мұны жасамаса, онда операциялық жүйенің бағдарламашысы барлық пиксельдердің электр өрістерін нақты бағдарламалауы керек еді. Егер контроллер болса, операциялық жүйе оны бірнеше параметрлер арқылы іске қосады, олардың ішінде жолдағы таңбалар немесе пиксельдер саны және экрандағы жолдар саны және контроллерге электр өрістерін басқаруға қамқорлық жасалады.

4.3 Жады кеңістігінде көрсетілген енгізу-шығару

Орталық процессормен байланысу үшін әр контроллерде бірнеше регистрлер бар. Осы регистрлерге жазу арқылы операциялық жүйе құрылғыға деректерді беру, деректерді қабылдау, қосу, өшіру немесе басқа әрекеттерді орындау туралы бұйрықтар бере алады. Осы регистрлердегі деректерді оқи отырып, операциялық жүйе құрылғының қазіргі жағдайы, жаңа команданы қабылдауға дайын екендігі және т. б. туралы біле алады.

Басқару регистрлерімен қоса көптеген құрылғыларда операциялық жүйе деректерді оқи алатын және оларды жаза алатын деректер буфері бар. Мысалы, компьютерлердің экранда пиксельдерді көрсетудің ең көп таралған әдісі бейне жадының болуын қамтамасыз етеді, ол іс жүзінде бағдарламалар немесе операциялық жүйе жаза алатын деректер буфері болып табылады.

Бірақ бұл жерде сұрақ туындайды: орталық процессор басқару регистрлерімен және осы құрылғылардың буферлерімен қалай байланысады? Екі балама бар. Олардың біріншісінде әр басқару регистріне 8 немесе 16 биттік бүтін сан болып табылатын енгізу-шығару портының нөмірі тағайындалады. Барлық енгізу-шығару порттарының жиынтығы әдеттегі пайдаланушы бағдарламаларының кіруінен қорғалған енгізу-шығару порттарының кеңістігін құрайды (оған тек операциялық жүйе қол жеткізе алады). Арнайы енгізу-шығару пәрмендерін қолдану, мысалы

```
IN REG,PORT
```

орталық процессор Port басқару регистріндегі деректерді санап, нәтижені өзінің регистрінде сақтай алады. Осы сияқты,

```
OUT PORT,REG
```

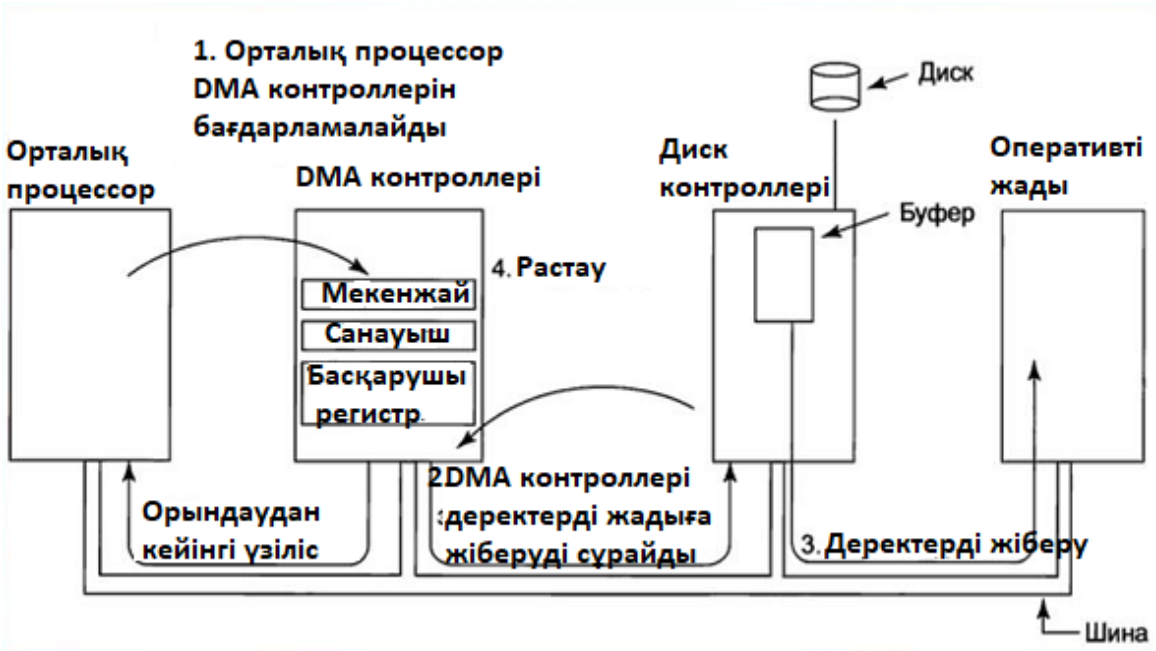
командасын қолдана отырып орталық процессор өз регистрінің мазмұнын PORT басқару регистріне жаза алады.

PDP-11 машиналарында пайда болған екінші нұсқа барлық басқару регистрлерін жады кеңістігіне көрсетуді қарастырады. Әрбір басқару регистріне жадта ерекше мекенжай бөлінген, ол жедел жадыда бөлінбейді. Бұл жүйе жадының мекенжай кеңістігіне енгізу-шығару деп аталады. Көптеген жүйелерде бөлінген мекенжайлар мекенжай кеңістігінің жоғарғы жағында немесе оның жанында орналасқан.

4.4 Жадыға тікелей қол жеткізу

Жады кеңістігінде көрсетілетін кіріс-шығыс процессорының болуына немесе болмауына қарамастан, олармен деректер алмасу үшін құрылғы контроллерлеріне хабарласу керек. Орталық процессор деректерді енгізу-шығару контроллерінен байт сұрай алады, бірақ оның жұмыс уақыты ұтымсыз жұмсалады, сондықтан жадыға тікелей қол жеткізу (Direct Memory Access (DMA)) деп аталатын басқа схема жиі қолданылады. Түсіндіруді қиындатпау үшін орталық процессор барлық құрылғыларға және жадыға орталық процессорды, жадыны және енгізу-шығару құрылғыларын қосатын бірыңғай жүйелік шина арқылы жүгінеді деп болжанады (8-сурет). Операциялық жүйе DMA-ны көптеген жүйелерде бар аппараттық DMA контроллері болған кезде ғана қолдана алады. Кейде бұл контроллер диск контроллерлеріне және басқа контроллерлерге енеді, бірақ бұл дизайн әр құрылғы үшін жеке DMA контроллерін қажет етеді. Көбінесе параллель режимде жиі өткізілетін бірнеше құрылғылармен деректер алмасуды реттеу үшін тек бір DMA контроллері қолжетімді (мысалы, жүйелік тақтаға орналастырылған).

8-суретте DMA контроллері физикалық түрде қай жерде болса да, орталық процессорға қарамастан жүйелік шинаға қолжетімді екендігі көрсетілген. Онда Орталық процессорға оқу және жазу үшін бірнеше регистрлер бар. Оларға жады мекен-жай регистрі, байт санауыш регистрі және бір немесе бірнеше басқару регистрі кіреді. Басқару тіркелімдерінде пайдаланылатын енгізу-шығару порты, деректерді беру бағыты (енгізу-шығару құрылғысынан оқу немесе оған жазу), берілетін ақпарат бірлігі (байттық немесе послабтық беру), сондай-ақ, бір пакетте берілетін байттар саны көрсетіледі.



8-сурет-DMA пайдалана отырып, деректерді беру кезінде жүзеге асырылатын операциялар

DMA жұмыс принципін түсіндіру үшін алдымен DMA қолданылмаған кезде дискіні қалай оқу керектігін қарастыру керек. Біріншіден, диск контроллері блокты (бір немесе бірнеше секторларды) дискіден бүкіл блок контроллердің ішкі буферінде болғанша дәйекті түрде оқиды. Содан кейін ол оқу қателерінің жоқтығына көз жеткізу үшін тексеру сомасын есептейді. Содан кейін контроллер үзілісті бастайды. Амалдық жүйе іске қосылған кезде, ол циклде контроллер буферінен диск блогын байт немесе сөз деп санай алады, циклдің әр өтуінде құрылғы контроллерінің тізілімінен бір байт немесе сөз оқып, оны жедел жадыда сақтай алады.

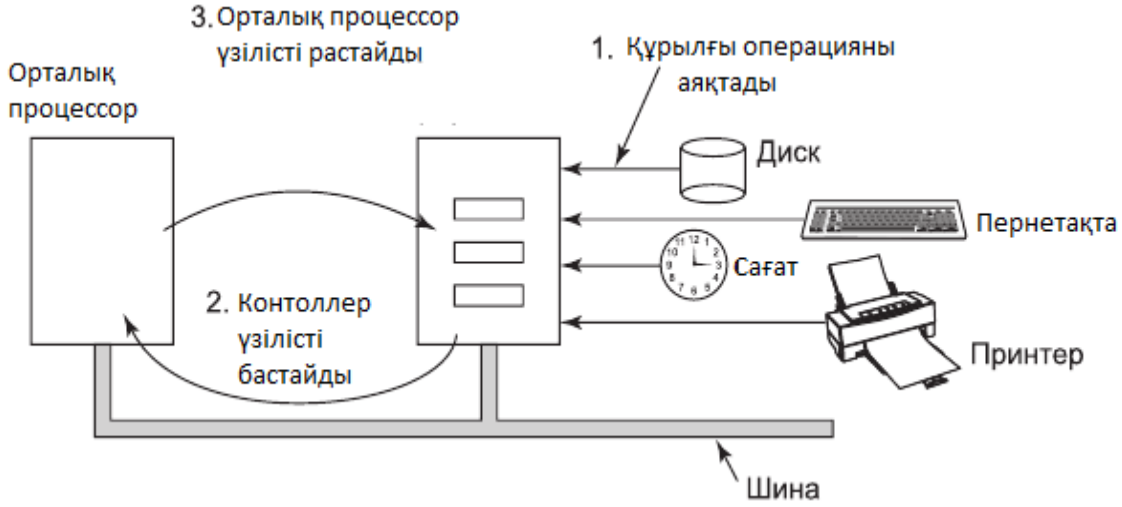
DMA қолданған кезде бәрі басқаша болады. Біріншіден, орталық процессор DMA контроллерін бағдарламалайды, оның регистрлерінің мәндерін не және қайда өту керектігін білетін етіп орнатады (8 суреттегі 1-қадам). Сондай-ақ, ол диск контроллеріне дискіден деректерді контроллердің ішкі буферіне оқуға және бақылау сомасын тексеруге бұйрық береді. Контроллер буферінде сенімді деректер болғаннан кейін, DMA жұмыс істей бастайды.

DMA контроллері диск контроллеріне шина арқылы оқу сұранысын беру арқылы деректерді беруді бастайды (2-қадам). Бұл оқу сұранысы кез келген басқа оқу сұранысымен бірдей көрінеді, ал диск контроллері оның қайдан келгенін білмейді немесе тіпті қызығушылық танытпайды - орталық процессордан ба әлде DMA контроллерінен бе. Әдетте, жазуға керекті жады мекенжайы шинаның мекенжай сызықтарына орнатылады, сондықтан диск

контроллері өзінің ішкі буферінен келесі сөзді алған кезде, оны қайда жазу керектігін біледі. Жадыға жазу - бұл тағы бір стандартты шина циклы (3-қадам). Жазу аяқталғаннан кейін диск контроллері де шина арқылы DMA контроллеріне растау сигналын жібереді (4-қадам). Содан кейін DMA контроллері пайдаланылатын жады мекенжайын көбейтеді және байт есептегішінің мәнін азайтады. Егер байт есептегіш әлі де нөлден үлкен болса, онда 2-ден 4-ке дейінгі қадамдар есептегіш мәні нөлге дейін қайталанады. Бұл орын алғаннан кейін, DMA контроллері орталық процессорға деректерді беру аяқталғанын білу үшін үзіліс жасайды. Операциялық жүйе жұмыс істей бастағанда, оның енді диск блогын жадыға көшірудің қажеті жоқ, өйткені ол қазірдің өзінде бар.

4.5 Үзілістер

Аппараттық деңгейде үзілістер келесідей жұмыс істейді. Енгізу-шығару құрылғысы өзіне тапсырылған жұмысты аяқтаған кезде, ол үзілісті бастайды (үзілістерге операциялық жүйе рұқсат берген жағдайда). Бұл сигналды арнайы бөлінген шина сызығына қою арқылы жасалады. Жүйелік тақтада орналасқан үзіліс контроллерінің чипі бұл сигналды анықтайды және келесі әрекеттердің сипаты туралы шешім қабылдайды. Типтік дербес компьютерлік жүйеде 9-суретте көрсетілген үзілістер құрылымы бар.



9-сурет - Үзілістің пайда болу тәртібі

Егер басқа кідірістер болмаса, үзіліс контроллері басталған үзілісті дереу өңдейді. Бірақ егер ол басқа үзілісті өңдеу процесінде болса немесе бір уақытта басқа құрылғы шинаға жоғары басымдық деңгейін үзу туралы сұрау

салса, онда құрылғы біраз уақытқа еленбейді. Бұл жағдайда ол орталық процессор қызмет еткенге дейін шинаға үзіліс сигналын қоюды жалғастырады.

Үзілісті өңдеу үшін контроллер мекенжай сызықтарына нөмір қояды, қай құрылғы назар аударуды қажет ететінін көрсетеді және орталық процессордың үзілуіне сигнал береді.

Үзіліс сигналы тоқтату әкеледі орталық процессор атқарып жатқан жұмысын тоқтатуына әкеліп, басқа жұмысқа кірісуіне әкеледі. Мекенжай жолдарындағы нөмір жаңа командалық санауыштың мәні алынған үзіліс векторы деп аталатын кестеде индекс ретінде қолданылады. Бұл командалық санауыш тиісті үзілісті өңдеу процедурасының басталуын көрсетеді. Әдетте, осы сәттен бастап жүйелік және әдеттегі үзілістер бірдей механизмді қолданады және көбінесе бірдей үзіліс векторын қолданады. Үзіліс векторының орналасқан жері машинаның өзінде «тігілген» болуы мүмкін немесе орталық процессор регистрімен көрсетілген жерде (операциялық жүйе жүктеген) жадыда болуы мүмкін.

Іске қосылғаннан кейін бірден үзілісті өңдеу процедурасы үзіліс контроллерінің кіріс-шығыс порттарының біріне белгілі бір мәнді жазып, үзілістің алынғанын растайды. Бұл растау контроллерге жаңа үзіліс жасай алатындығын хабарлайды. Орталық процессор жаңа үзілісті өңдеуге дайын болғанға дейін бұл растауды кешіктірудің арқасында бірнеше (бір уақытта дерлік шығарылатын) үзілістердің болуына байланысты бәсекелестікті жоюға болады. Айтпақшы, кейбір ескірген компьютерлерде орталықтандырылған үзіліс контроллері жоқ, сондықтан әр құрылғы контроллері өзінің үзілістерін орнатады.

Техникалық қызмет көрсету процедурасын бастамас бұрын, жабдық әрқашан белгілі бір ақпаратты сақтайды. Сақталатын ақпарат және оны сақтау орны процессордан процессорға дейін әр түрлі болады. Үзілген процесті жалғастыру үшін кем дегенде командалық санауышты сақтау керек. Бағдарламалық қамтамасыз етудің барлық регистрлерін және орталық процессордың көптеген ішкі регистрлерін сақтау тағы бір шекті жағдай болады.

Сұрақ туындайды: бұл ақпаратты қайда сақтау керек? Мәнін операциялық жүйе қажетіне қарай оқи алатын ішкі регистрлерде орналастыруға болады. Бірақ бұл жағдайда барлық ықтимал маңызды ақпарат оқылғанға дейін үзіліс контроллеріне растау беруге мүмкіндік бермейтін мәселе туындайды, өйткені күйді сақтау кезінде келесі үзіліс барлық ішкі регистрлерді қайта жазады. Мұндай стратегия үзілістерге тыйым салынған ұзақ үзілістерге және үзіліс сигналдарының ықтимал жоғалуына және деректердің жоғалуына әкеледі.

Сондықтан көптеген орталық процессорлар ақпаратты стекте сақтайды. Бірақ бұл тәсілдің де өз проблемалары бар. Біріншіден, сұрақ туындайды: деректерді қай стекте сақтау керек? Егер сіз ағымдағы стекті қолдансаңыз, ол пайдаланушы процесінің стегі болуы мүмкін. Стек көрсеткішінде тіпті жарамсыз мән болуы мүмкін, бұл жабдық көрсетілген мекенжайға бірнеше сөз жазуға тырысқанда фатальды қатеге әкеледі. Ол сонымен қатар, беттің соңын көрсете алады. Бірнеше жазбадан кейін ол шегінен шығуы мүмкін және беттің болмау қатесі пайда болады. Аппараттық үзілісті өңдеу кезінде бұл қатенің пайда болуы өте күрделі мәселені тудырады: беттің жоқтығының қатесін өңдеу үшін күйді қайда сақтау керек?

Негізгі стекті пайдалану кезінде стек көрсеткішінің рұқсат етілген мәні бар және бекітілген бетті көрсететін әлдеқайда жоғары ықтималдығы бар. Бірақ ядро режиміне ауысу MMU контекстін өзгертуді қажет етуі мүмкін және кэш пен TLB мазмұнының көп бөлігін немесе тіпті барлығын жарамсыз етуі мүмкін. Олардың статикалық немесе динамикалық қайта жүктелуі үзілісті өңдеу уақытын арттырады және процессордың уақытын ысырап етеді.

4.6 Құрылғы драйверлері

Әр контроллерде оған пәрмендер жіберуге арналған құрылғы регистрлері немесе оның күйін оқу үшін қолданылатын құрылғы регистрлері немесе сол және басқа регистрлер бар. Құрылғы регистрлерінің саны мен командалардың сипаты нақты құрылғыға байланысты өте ерекшеленеді. Мысалы, тінтуір драйвері тінтуірден оның қаншалықты жылжытылғанын және қазіргі уақытта қандай түймелер босатылғанын көрсететін ақпаратты қабылдауы керек. Керісінше, диск драйвері секторлар, жолдар, цилиндрлер, бастар, бастар блогының қозғалысы, электр жетектері, бастың тұрақтануының уақытша көрсеткіштері және дискінің қалыпты жұмысын қамтамасыз ететін барлық басқа механизмдер туралы білуі керек. Әрине, бұл драйверлер бір-бірінен мүлдем өзгеше болады.

Сондықтан компьютерге қосылған әрбір енгізу-шығару құрылғысын басқару үшін оның ерекшеліктерін ескеретін арнайы бағдарлама қажет. Бұл бағдарлама құрылғы драйвері деп аталады. Әдетте оны құрылғы өндірушісі жасайды және осы құрылғымен бірге келеді. Әр операциялық жүйеге жеке драйверлер қажет болғандықтан, құрылғы өндірушісі әдетте бірнеше танымал операциялық жүйелер үшін драйверлерді жеткізеді.

Құрылғының әр драйвері әдетте құрылғының бір түрін немесе байланысты құрылғылардың ең көп дегенде бір класын басқарады. Мысалы, SCSI диск драйвері әдетте әртүрлі көлемдегі және әртүрлі жылдамдықтағы бірнеше SCSI дискілерін басқара алады, мүмкін SCSI интерфейсі бар Blu-ray

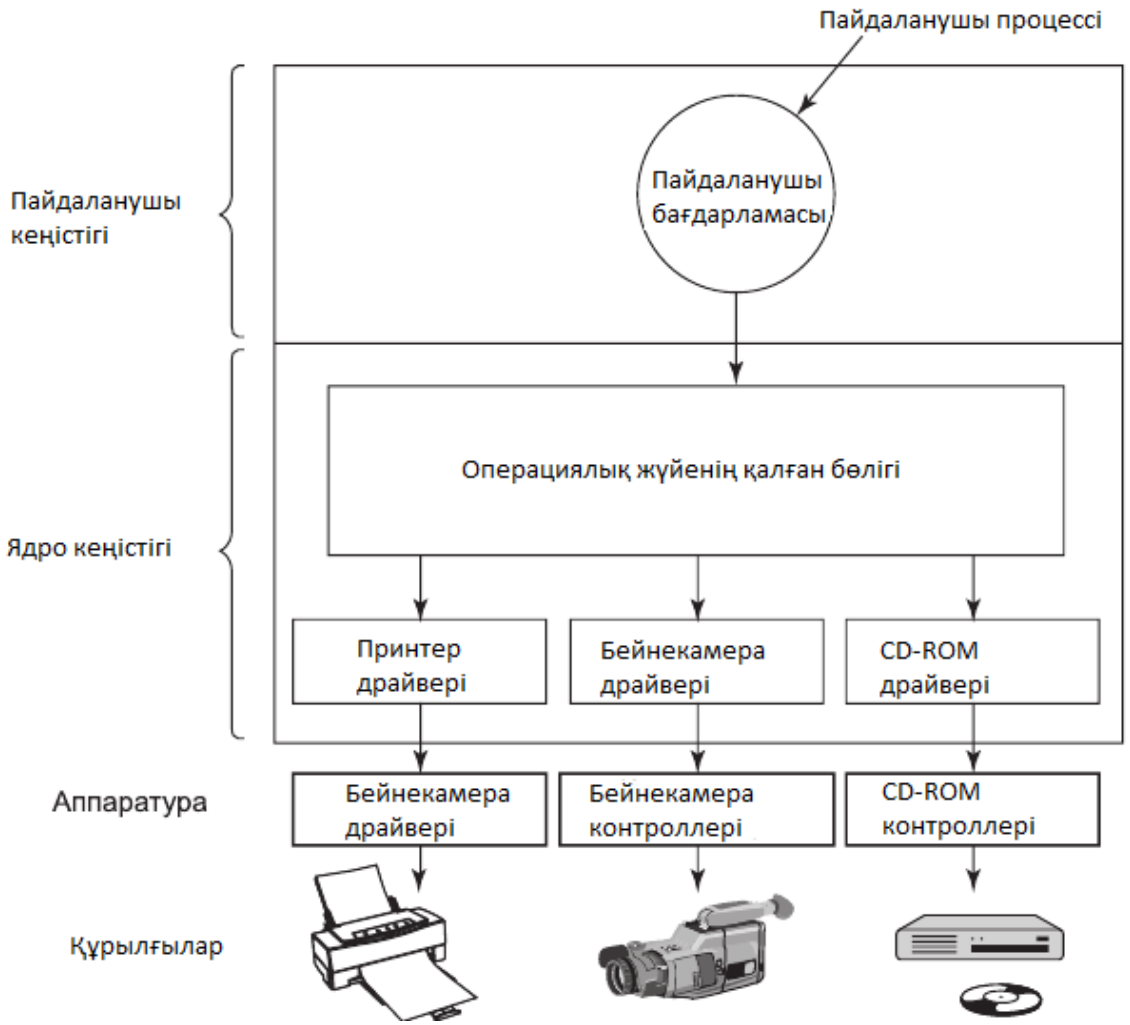
дискісі. Бірақ тышқандар мен Джойстиктер бір-бірінен соншалықты ерекшеленеді, олар әдетте әртүрлі драйверлерді қажет етеді. Дегенмен, бірнеше гетерогенді құрылғыларды басқаратын бір құрылғы драйверін құру техникалық тұрғыдан мүмкін. Бірақ бұл көп жағдайда ең жақсы идея емес.

Бірақ кейде мүлдем басқа құрылғылар бірдей негізгі технологияға негізделген. Мүмкін, ең танымал мысал сериялық шинаның USB технологиясы болуы мүмкін және оның әмбебап деп аталатыны бекер емес. USB құрылғыларына дискілер, жады карталары, камералар, тышқандар, пернетақталар, шағын желдеткіштер, сымсыз желі карталары, Роботтар, несие карталарын оқитындар, батарея ұстаралары, қағаз ұсатқыштар, штрих-код сканерлері және портативті термометрлер кіреді., Олар мүлдем әр түрлі нәрселермен айналысқандарымен олардың барлығы USB қолданады. Ерекшелігі - USB драйверлерінен әдетте желілерде TCP/IP стек тәрізді стек пайда болады. Төменгі жағында, әдетте, жабдықта USB-сілтемелер деңгейі (сериялық енгізу-шығару) бар, ол сигнализация және сигнал ағынын USB пакеттеріне декодтау сияқты жабдыққа қатысты барлық нәрсені өңдейді. Ол көптеген құрылғыларға ортақ USB пакеттері мен мүмкіндіктерімен жұмыс істейтін жоғары деңгейлер үшін қолданылады. Ақыр соңында, жоғарыда жоғары деңгейлі API бар, мысалы, дискілер, камералар және т.б. Осылайша, хаттама стегінің бөліктерін ортақ пайдаланғанына қарамастан, бізде әлі де бөлек құрылғы драйверлері бар.

Құрылғының аппараттық құралына, яғни контроллер регистрлеріне қол жеткізу үшін құрылғы драйвері әдетте операциялық жүйенің ядросының бөлігі болуы керек, кем дегенде қазіргі архитектураларда. Бірақ, шын мәнінде, құрылғы регистрлерін оқу және жазу үшін жүйелік шақыруларды қолдана отырып, пайдаланушы кеңістігінде жұмыс істейтін драйверлерді жасауға болады. Бұл шешім ядроны драйверлерден және драйверлерді бір-бірінен оқшаулауға мүмкіндік береді, ал жүйенің сәтсіздіктерінің негізгі көзі - ядроның жұмысына кедергі келтіретін «шикі» драйверлерді жояды. Әрине, бұл жоғары сенімді жүйелерді құру кезінде жақсы шешім. Құрылғы драйверлері пайдаланушы процестері ретінде жұмыс істейтін жүйенің мысалы MINIX 3 (www.minix3.org). Бірақ басқа жұмыс үстелі операциялық жүйелерінің көпшілігі драйверлерді өзекке қосуды ұсынатындықтан, дәл осындай модель қарастырылады.

Кез-келген операциялық жүйені жасаушылар басқа әзірлеушілер жасаған бағдарламалық кодтың бөліктері (драйверлер) олардың жүйесіне орнатылатындығын білетіндіктен, оларға осындай орнатуға мүмкіндік беретін архитектура қажет. Бұл дегеніміз, драйвердің не істейтіні және оның бүкіл операциялық жүйемен қалай әрекеттесетіні туралы нақты модель болуы

керек. 10-суретте көрсетілгендей, құрылғы драйверлері әдетте операциялық жүйенің қалған компоненттерінен төмен орналастырылады.



10-сурет - Құрылғы драйверлерінің логикалық орналасуы

Әдетте, операциялық жүйе драйверлерді бірнеше санаттардың біріне жатқызады. Ең көп таралған санаттар - блок құрылғыларының драйверлері, оларға барлық басқа блоктарға тәуелсіз қол жеткізуге болатын көптеген деректер блоктары бар диск драйверлері және пернетақта мен принтер драйверлерін қамтитын символдық құрылғы драйверлері - таңбалар ағынын құратын немесе қабылдайтын құрылғылар кіреді.

Көптеген операциялық жүйелер үшін стандартты интерфейс анықталған, оны блок құрылғыларының барлық драйверлері қолдауы керек және символдық құрылғылардың барлық драйверлері қолдауы керек тағы бір стандартты интерфейс анықталған. Бұл интерфейстер бірнеше

процедуралардан тұрады, оларды драйверге қол жеткізу үшін операциялық жүйенің қалған бөлігі шақыра алады. Бұл процедураларға, мысалы, блокты оқу (блок құрылғыларында) немесе таңбалар жолын жазу (символдық құрылғыларда) процедуралары жатады.

Кейбір жүйелерде операциялық жүйе екілік кодтарда бірыңғай бағдарлама болып табылады, онда оған қажетті барлық құрастырылған драйверлер бар. Мұндай схема көптеген жылдар бойы UNIX отбасылық жүйелері үшін норма болды, өйткені олар енгізу-шығару құрылғылары өте сирек өзгеретін компьютерлік орталықтарда жұмыс істеді. Жаңа құрылғыны қосқан кезде, жүйелік администратор жаңа екілік кодты жасау үшін жаңа драйвері бар ядроны қайта компиляциялады.

Көптеген енгізу-шығару құрылғылары бар дербес компьютерлер дәуірінің басталуымен бұл модель енді жұмыс істемейді. Аз ғана пайдаланушылар бастапқы кодтары немесе объект модульдері болса да, өзегін қайта құрастыруға немесе қайта құрастыруға қабілетті, бұл өте сирек кездеседі. Оның орнына, MS-DOS-дан бастап операциялық жүйелер драйверлер жұмыс барысында жүйеге динамикалық түрде жүктеле бастаған модельге көшті. Драйверді жүктеуді басқару әртүрлі жүйелерде әртүрлі жүзеге асырылады.

Құрылғы драйверіне бірнеше функция жүктелген. Олардың ішіндегі ең айқындары - белгілі бір құрылғыларға тәуелсіз бағдарламалық жасақтаманың оқу және жазу туралы дерексіз сұраныстарын қабылдау және олардың орындалу тәртібін бақылау. Бірақ оларға бірқатар басқа функциялар жүктелген. Мысалы, драйвер қажет болған жағдайда құрылғыны іске қосуы керек. Сондай-ақ, құрылғының қуат тұтынуын басқару және оқиғаларды тіркеу үшін қажет болуы мүмкін.

Көптеген құрылғы драйверлері ұқсас жалпы құрылымға ие. Әдеттегі драйвер өз жұмысын кіріс параметрлерінің жарамдылығын тексеруден бастайды. Егер олар қолайсыз болса, қате туралы хабарлама қайтарылады. Егер параметрлер жақсы болса, дерексіз ұғымдарды нақты ұғымдарға аудару қажет болуы мүмкін. Диск драйвері үшін бұл қалыпты блок нөмірін диск геометриясына қатысты бас, жол, сектор және цилиндр нөмірлеріне түрлендіруді білдіруі мүмкін.

Содан кейін драйвер құрылғының қазіргі уақытта қолданылып жатқанын тексере алады. Егер ол пайдаланылса, сұрау кейінгі өңдеу үшін кезекке қойылады. Егер құрылғы тоқтап қалса, сұранымның өңделетінін анықтау үшін жабдықтың жағдайы тексеріледі. Деректерді беруді бастамас бұрын, құрылғыны қосу немесе оның қозғалтқышын іске қосу қажет болуы мүмкін. Құрылғы қосылғаннан кейін ол жұмысқа дайын болады, оны басқаруға болады.

Құрылғыны басқару дегеніміз - оның мекен-жайына командалар тізбегін беру. Бұл не істеу керек екеніне байланысты командалар тізбегін анықтайтын драйвер. Драйвер қандай командаларды шығаратындығын түсінгеннен кейін, оларды құрылғы контроллерінің регистрлеріне жаза бастайды. Әр команданы контроллерге жазғаннан кейін контроллердің команданы қабылдағанын және келесі команданы қабылдауға дайын екенін тексеру қажет болуы мүмкін. Бұл дәйектілік барлық командалар шығарылғанға дейін қайталанады. Кейбір контроллерлерге байланысты командалар тізімін (жадыда) көрсетуге болады және операциялық жүйенің көмегінсіз осы командаларды өздігінен оқып, өңдеуді бұйырады.

Командалар шығарылғаннан кейін екі жағдайдың бірі болуы мүмкін. Көп жағдайда драйвер контроллер қандай да бір жұмысты өз пайдасына жасағанша күтуі керек, сондықтан оны блоктан босату үшін үзіліс болғанша ол өзін-өзі блоктайды. Бірақ басқа жағдайларда операция кідіріссіз аяқталады және драйверді блоктаудың қажеті жоқ. Соңғы жағдайдың мысалы ретінде бірнеше байтты контроллер регистрлеріне жазуды қажет ететін символдық режимде экранды айналдыруға болады. Мұны істеу үшін ешқандай механикалық қозғалыс қажет емес, сондықтан бүкіл операцияны бірнеше наносекундта аяқтауға болады.

Бірінші жағдайда, блокталған драйвер үзіліс арқылы іске қосылады. Екінші жағдайда, ол ешқашан белсенді емес күйге өтпейді. Қалай болғанда да, операция аяқталғаннан кейін драйвер қателіктердің болмауын тексеруі керек. Егер барлығы жақсы болса, драйвер қолданылатын құрылғыға тәуелді емес бағдарламалық жасақтамаға беру үшін деректерді (мысалы, жай ғана оқылған блок) ала алады. Соңында, ол оны тудырған бағдарламаға құрылғының күйі, қателердің болуы немесе болмауы туралы белгілі бір ақпаратты қайтарады. Егер кезекте басқа сұраулар болса, енді олардың біреуін таңдап, іске қосуға болады. Егер кезекте сұраулар болмаса, келесі сұрауды күту кезінде драйвер блокталады.

Бұл жеңілдетілген модель нақты жұмыс туралы өте жуық түсінік береді. Шын мәнінде, көптеген факторларға байланысты бағдарламалық код әлдеқайда күрделі. Ең алдымен енгізу-шығару құрылғысы өз жұмысын және драйвер жұмысын тоқтатуы мүмкін. Үзіліс драйверді іске қосуы мүмкін. Айтпақшы, ол ағымдағы драйверді іске қосуы мүмкін. Мысалы, кіріс пакеттің желілік драйверін өңдеу кезінде тағы бір пакет келуі мүмкін. Сондықтан драйверлер қайта жұмыс істеуі керек, яғни жұмыс істейтін драйвер бірінші шақыру аяқталғанға дейін қайта шақыруды күтуі керек.

Ыстық қосуға жол беретін жүйеде құрылғы компьютердің жұмыс істеп тұрған кезінде қосылуы немесе шығарылуы мүмкін. Нәтижесінде, драйвер кез-келген құрылғыдан оқумен айналысып жатқанда, жүйе оған пайдаланушы

бұл құрылғыны жүйеден кенеттен алып тастағанын айта алады. Драйвер ядро деректер құрылымына зақым келтірместен ағымдағы деректерді беруді тоқтатып қана қоймай, сонымен бірге, жүйеден жаңадан шығарылған құрылғыға қатысты барлық сұраныстарды алып тастауға және оларды жіберген бағдарламаларға жағымсыз жаңалықтарды хабарлауға мәжбүр болады. Сонымен қатар, жаңа құрылғылардың күтпеген қосылуы ядроға драйверден ескілерін алып, орнына жаңаларын беру арқылы ресурстарды қайта бөлуге мәжбүр етуі мүмкін (мысалы, үзілістерді сұрау сызықтары).

Драйверлерге жүйелік шақырулар жасауға рұқсат етілмейді, бірақ көбінесе ядроның қалған бөлігімен өзара әрекеттесу қажеттілігі туындайды. Әдетте оларға белгілі бір ядро процедураларын шақыруға рұқсат етіледі. Мысалы, буфер ретінде арнайы жады беттерін пайдалану үшін драйверлер оларды бөлумен және босатумен айналысатын процедураларды тудырады. MMU, таймерлер, DMA контроллері, үзіліс контроллері және т.б. басқару үшін басқа пайдалы шақырулар қажет

5 ФАЙЛДЫҚ ЖҮЙЕ

5.1 Файлдық жүйенің негізгі функциялары

Барлық компьютерлік қосымшалар ақпаратты сақтау және алу керек. Жеке мекенжай кеңістігінде жұмыс процесі шектеулі деректерді сақтай алады, ал мұндай сақтау сыйымдылығы виртуалды мекенжай кеңістігінің көлемімен шектеледі. Кейбір қосымшалар үшін бұл өлшем жеткілікті, бірақ басқалары үшін, мысалы, әуе билеттерін брондау жүйелері, банктік немесе корпоративтік есепке алу жүйелері үшін виртуалды мекенжай кеңістігі жеткіліксіз болады.

Осылайша, ақпаратты ұзақ мерзімді сақтау құрылғыларына келесі үш маңызды талап қойылады:

- құрылғылар үлкен көлемде деректерді сақтауы керек;
- ақпарат процесс тоқтатылғаннан кейін сақталуы керек;
- процестер ақпаратқа параллель қолжетімді болуы керек.

Барлық осы мәселелердің әдеттегі шешімі - ақпаратты файл деп аталатын модульдерге дискілерге және басқа сыртқы сақтаушыларға орналастыру. Процестер қажет болған жағдайда оларды оқып, жаңа файлдар жасай алады. Файлдарда сақталған ақпарат тұрақты болуы керек. Файл иесі файлды жою пәрменін берген кезде ғана жоғалып кетуі керек.

Файлдарды операциялық жүйе басқарады. Олардың құрылымы, атауы, қолданылуы, қорғалуы, іске асырылуы және қолжетімділігі операциялық жүйенің маңызды элементтері болып табылады. Файлдармен жұмыс істейтін операциялық жүйенің бөлігі файлдық жүйе деп аталады.

Файл - бұл деректерді жазуға және оқуға болатын сыртқы жадының белгілі аймағы. Файлдар қуатқа тәуелсіз жадыда сақталады, әдетте магниттік дискілерде.

Файлды пайдаланудың негізгі мақсаттары:

- ақпаратты ұзақ және сенімді сақтау. Ұзақ мерзімділікке қуатқа тәуелсіз сақтау құрылғыларын пайдалану арқылы қол жеткізіледі, ал жоғары сенімділік файлдарға қол жеткізуді қорғау құралдарымен және ОЖ бағдарламалық кодының жалпы ұйымымен анықталады, онда жабдықтың істен шығуы көбінесе файлдарда сақталатын ақпаратты жоймайды;
- ақпаратты бөлісу. Файлдар адамға түсінікті символдық атаудың болуына және сақталған ақпараттың тұрақтылығы мен файлдың орналасуына байланысты қосымшалар мен пайдаланушылар арасында ақпаратты бөлудің табиғи және қарапайым әдісін ұсынады. Пайдаланушыда файлдарды топтарға біріктіретін анықтамалық каталогтар, белгілер бойынша файлдарды іздеу құралдары, файлдарды құруға, өзгертуге және жоюға арналған командалар

жиынтығы бар файлдармен жұмыс істеудің ыңғайлы құралдары болуы керек. Файлды бір пайдаланушы жасай алады, содан кейін оны мүлдем басқа пайдаланушы қолдана алады, ал файл жасаушы немесе администратор оған басқа пайдаланушылардың кіру құқығын анықтай алады. Бұл мақсаттар ОЖ-де файлдық жүйемен жүзеге асырылады.

Файлдық жүйе (ФЖ) - бұл операциялық жүйенің бөлігі:

- дискідегі барлық файлдардың жиынтығы;

- файлдарды басқару үшін пайдаланылатын деректер құрылымдарының жиынтығы;

- кешені жүйелік бағдарламалық құралдар үшін файлдармен операциялар жүргізу.

Файлдық жүйе бағдарламаларға файлды білдіретін кейбір дерексіз объектіге әрекеттерді орындау үшін жеткілікті қарапайым операциялар жиынтығын басқаруға мүмкіндік береді. Бұл жағдайда бағдарламашыларға дискідегі деректердің нақты орналасуы, деректерді буферлеу және ұзақ мерзімді сақтау құрылғысынан деректерді берудің басқа да төмен деңгейлі мәселелері туралы егжей-тегжейлі мәліметтің қажеті жоқ. Файлдық жүйе осы функциялардың барлығын өзіне алады. Файлдық жүйе дискіні таратады, файл атауын қолдайды, файл атауларын сыртқы жадтағы тиісті мекенжайларға көрсетеді, деректерге қол жеткізуді қамтамасыз етеді, файлдарды бөлуді, қорғауды және қалпына келтіруді қолдайды.

Осылайша, файлдық жүйе ұзақ мерзімді деректер қоймасын физикалық ұйымдастырудың барлық қиындықтарын қорғайтын және бағдарламалар үшін осы сақтаудың қарапайым логикалық моделін құратын, сонымен қатар оларға файлдарды басқаруға ыңғайлы командалар жиынтығын беретін аралық қабат рөлін атқарады.

ФЖ шешетін міндеттер есептеу процесін тұтастай ұйымдастыру әдісіне байланысты. ФЖ-дегі негізгі функциялар келесі міндеттерді шешуге бағытталған:

- файлдар атауы;

- қосымшаларға арналған бағдарламалық интерфейс;

- деректер қоймасын физикалық ұйымдастыруға файлдық жүйенің логикалық моделін көрсету;

- файлдық жүйенің қуат ақауларына, аппараттық және бағдарламалық қамтамасыз ету қателеріне тұрақтылығы.

ФЖ міндеттері бір пайдаланушының жұмыс істеуіне арналған, бірақ бір уақытта бірнеше процестерді іске қосуға мүмкіндік беретін бір пайдаланушының көп бағдарламалы операциялық жүйелерінде күрделене түседі. Осы типтегі алғашқы ОЖ-нің бірі OS/2 болды. Жоғарыда аталған тапсырмаларға бірнеше процестерден файлға жаңа ортақ тапсырма қосылады.

Бұл жағдайда файл ортақ ресурс болып табылады, яғни файлдық жүйе осындай ресурстарға қатысты мәселелердің бүкіл кешенін шешуі керек. Атап айтқанда, ФЖ-де файлды және оның бөліктерін блоктау, жарыстарды болдырмау, тұйықтарды жою, көшірмелерді келістіру және т.б. құралдары көзделуі тиіс.

Бірнеше пайдаланушы жүйелерінде тағы бір міндет пайда болады: бір пайдаланушының файлдарын басқа пайдаланушының рұқсатынсыз кіруінен қорғау.

Желілік ОЖ құрамында жұмыс істейтін ФЖ функциялары одан да күрделі болады. Бұл тақырып желілік ресурстарды басқаруға арналған кітаптың соңғы тарауында қарастырылады.

Басқаша айтқанда, «файлдық жүйе» термині, ең алдымен, файлдарға ұйымдастырылған деректерге қол жеткізу принциптерін анықтайды. Дәл осындай термин көбінесе белгілі бір деректер тасымалдаушысында орналасқан нақты файлдарға қатысты қолданылады. «Файлдарды басқару жүйесі» термині файлдық жүйенің нақты іске асырылуына қатысты қолданылуы керек, яғни бұл белгілі бір операциялық жүйеде файлдармен жұмыс істеуді қамтамасыз ететін бағдарламалық модульдер жиынтығы.

Тағы да ййта кету керек, кез-келген файлды басқару жүйесі өздігінен жоқ - ол белгілі бір ОЖ-де жұмыс істеуге арналған. Мысал ретінде, әйгілі FAT файлдық жүйесі (file allocation table) файлдарды басқару жүйесі ретінде көптеген енгізулерге ие деп айтуға болады. Сонымен, бұл атауды алған және алғашқы дербес компьютерлер үшін жасалған жүйе жай FAT деп аталды (қазір ол FAT-12 деп аталады). Ол дискеталармен жұмыс жасау үшін жасалынған және біраз уақыт қатты дискілермен жұмыс жасау кезінде қолданылған. Содан кейін ол үлкен көлемді қатты дискілермен жұмыс істеу үшін жетілдірілді және бұл жаңа іске асыру FAT-16 деп аталды. Біз бұл файлдық жүйенің атауын MS-DOS файлдарды басқару жүйесіне қатысты қолданамыз. Fat жүйесінің негізгі принциптерін қолданатын OS/2 үшін файлдарды басқару жүйесін енгізу super-FAT деп аталады; басты айырмашылық - әр файл үшін кеңейтілген атрибуттарды сақтау мүмкіндігі. FAT және Windows 95/98, Windows NT және т. б. үшін файлдарды басқару жүйесінің нұсқасы бар. Басқаша айтқанда, кейбір файлдық жүйеге сәйкес ұйымдастырылған файлдармен жұмыс істеу үшін әр ОЖ үшін тиісті файлдарды басқару жүйесі жасалуы керек. Бұл файлдарды басқару жүйесі тек ол жасалған ОЖ-де жұмыс істейді; бірақ сонымен бірге ол файлдық жүйенің бірдей негізгі принциптері бойынша жұмыс істейтін басқа ОЖ-нің файлдарды басқару жүйесі арқылы жасалған файлдармен жұмыс істеуге мүмкіндік береді.

5.2 Файлдық жүйенің компоненттері

Файлдар дерексіз механизмге жатады. Олар ақпаратты дискіге сақтап, кейінірек қайта оқуға мүмкіндік береді. Сонымен қатар, пайдаланушыдан ақпаратты сақтау әдісі мен орны, сондай-ақ, дискілердің жұмысы туралы мәліметтер жасырылуы керек.

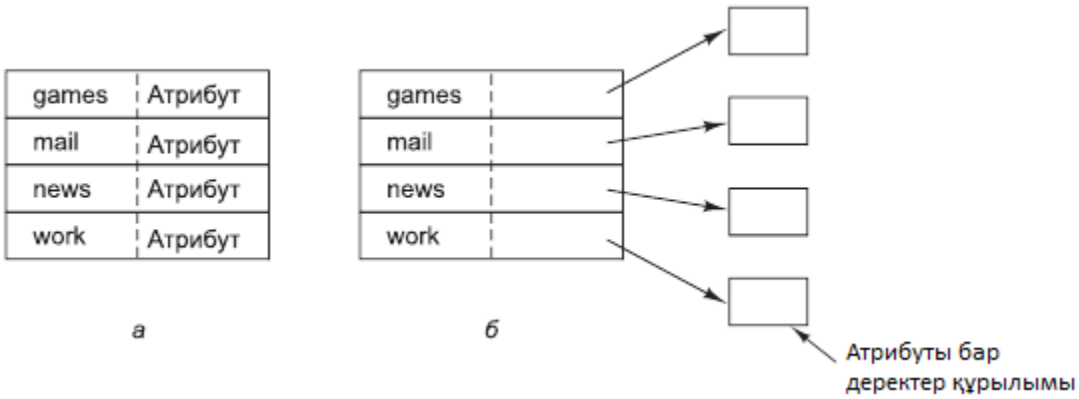
Мүмкін кез-келген дерексіз механизмнің маңызды сипаттамасы - басқарылатын нысандар қалай аталады, сондықтан біз файлдық жүйені файл атауынан бастаймыз. Файлды жасау кезінде процесс файлға атау береді. Процесс аяқталған кезде файл өмір сүруін жалғастырады және оның аты бойынша басқа процестер оған қол жеткізе алады.

Көптеген операциялық жүйелер әртүрлі файл түрлерін қолдайды. Мысалы, UNIX және Windows жүйелерінде әдеттегі файлдар мен каталогтар арасында айырмашылық бар. UNIX жүйесінде символдық және блоктық арнайы файлдар да ерекшеленеді. Кәдімгі (regular) файлдарға пайдаланушының ақпараты бар барлық файлдар кіреді. Каталогтар - бұл файлдық жүйені құрылымдауды қамтамасыз ететін жүйелік файлдар. Біз оларды төменде толығырақ қарастырамыз. Таңбаның арнайы файлдары кіріс/шығыс файлдарымен байланысты және терминалдар, принтерлер және желілер сияқты сериялық кіріс/шығыс құрылғыларын модельдеу үшін қолданылады. Блоктық арнайы файлдар дискілерді модельдеуде қолданылады. Стандартты файлдар негізінен ASCII жолдарының тізбегі немесе екілік байт жиынтығы болып табылады. Кейбір жүйелерде әр ASCII жолы арбаны қайтару символымен аяқталады. Басқаларында (мысалы, UNIX) жолды аудару символы қолданылады. Екі таңба қолданылатын жүйелер бар (мысалы, MS-DOS). Жолдардың ұзындығы бірдей болуы міндетті емес.

Файлдар ақпаратты сақтауға және оны кейінірек алуға мүмкіндік береді. Әртүрлі операциялық жүйелерде әртүрлі файлдық операциялар жиынтығы бар.

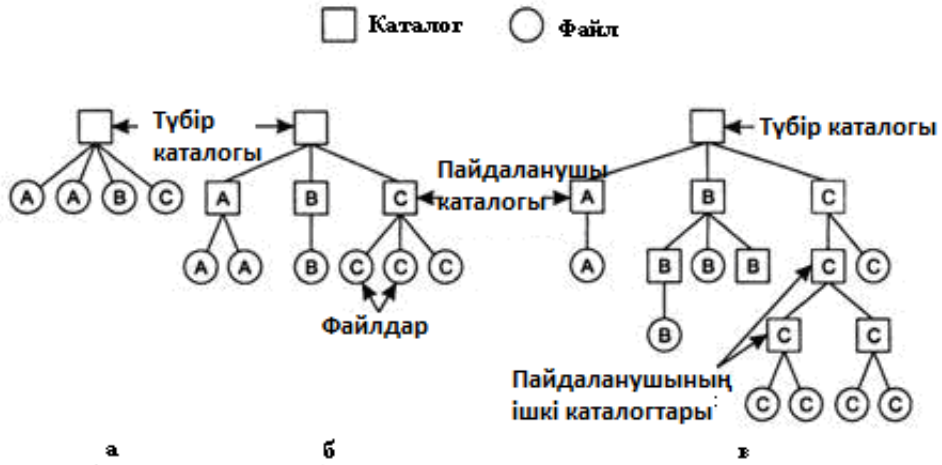
Файлдық жүйелерде файлдар әдетте каталогтарға ұйымдастырылады, олар өз кезегінде көптеген операциялық жүйелерде де файлдар болып табылады. Бұл бөлімде біз каталогтарды, олардың ұйымдастырылуын, қасиеттерін және олардың үстінен орындалатын әрекеттерді қарастырамыз.

Әдетте каталогта бірнеше жазбалар бар, әр файлға бір жазба. Опциялардың бірі 11-суретте көрсетілген, онда каталогтың әр жазбасына файл атауы, оның атрибуттары және дискідегі файл деректерінің мекен-жайы кіреді. Тағы бір нұсқа 11, б-суретте көрсетілген. Мұнда каталог жазбасы файл атауын және атрибуттары мен дискілік мекенжайлары бар деректер құрылымына меңзерді сақтайды. Осы екі нұсқа да кеңінен қолданылады.



11-сурет - Каталог: а - атрибуттар каталогта сақталады;
б - атрибуттар бөлек сақталады

Бұл идеяның дамуы екі деңгейлі иерархия болып табылады, онда әр пайдаланушыға жеке каталог бөлінеді (12, б-сурет).



12-сурет - Файлдық жүйенің үш нұсқасы:
а - барлық пайдаланушыларға бір каталог;
б - әр пайдаланушы үшін жеке каталог;
в - әр пайдаланушы үшін каталог ағашы. Әріптер каталогтың немесе файлдың иелерін көрсетеді

Екі деңгейлі иерархияның арқасында әртүрлі пайдаланушылар арасындағы файл атауларының қайшылықтары жоғалады, бірақ бұл көптеген файлдары бар пайдаланушылар үшін жеткіліксіз. Әдетте пайдаланушылар өз файлдарын логикалық түрде топтастыруы керек. Мысалы, профессорда бір курс үшін жазған кітабын құрайтын файлдар жиынтығы, басқа курс студенттеріне арналған бағдарламалары бар басқа файлдар болуы мүмкін.

Файлдардың үшінші жиынтығында профессор жасаған жаңа компилятордың бастапқы мәтіндері, файлдардың төртінші тобы - әртүрлі гранттардың ұсыныстары, сонымен қатар электрондық пошта, кездесулер кестесі, мақалалар, ойындар және т.б. болуы мүмкін.

Сондықтан бізге белгілі бір жалпы иерархия қажет (яғни каталог ағашы). Бұл тәсіл 12, в-суретте көрсетілген. Мұнда түбірлік каталогқа салынған A, B және C каталогтары әртүрлі пайдаланушыларға тиесілі, олардың екеуі жұмыс істейтін жобалар үшін ішкі каталогтар жасады. Ішкі каталогтардың еркін санын құру мүмкіндігі пайдаланушыларға өз жұмысын ұйымдастыруға мүмкіндік беретін қуатты құрылымдық құрал болып табылады. Осы себепті қазіргі заманғы файлдық жүйелердің барлығы дерлік осылай жүзеге асырылады.

Файлдық жүйені каталог ағашы ретінде ұйымдастырған кезде файлды көрсетудің белгілі бір әдісі қажет. Ол үшін әдетте екі түрлі әдіс қолданылады. Бірінші жағдайда файлға жүгіну түбірден файлға дейінгі барлық каталогтардың аттарынан және файлдың өзі атауынан тұратын жолдың абсолютті атауы бойынша орындалады.

Жолдың салыстырмалы атауы да қолданылады. Ол жұмыс каталогы тұжырымдамасымен бірге қолданылады (қазіргі каталог деп те аталады). Пайдаланушы каталогтардың біреуін ағымдағы жұмыс каталогына тағайындай алады. Бұл жағдайда бөлгіштің бастапқы таңбасы жоқ жолдардың барлық атаулары салыстырмалы болып саналады және ағымдағы каталогқа қатысты есептеледі.

6 ЖАДЫНЫ БАСҚАРУ

6.1 Свопинг

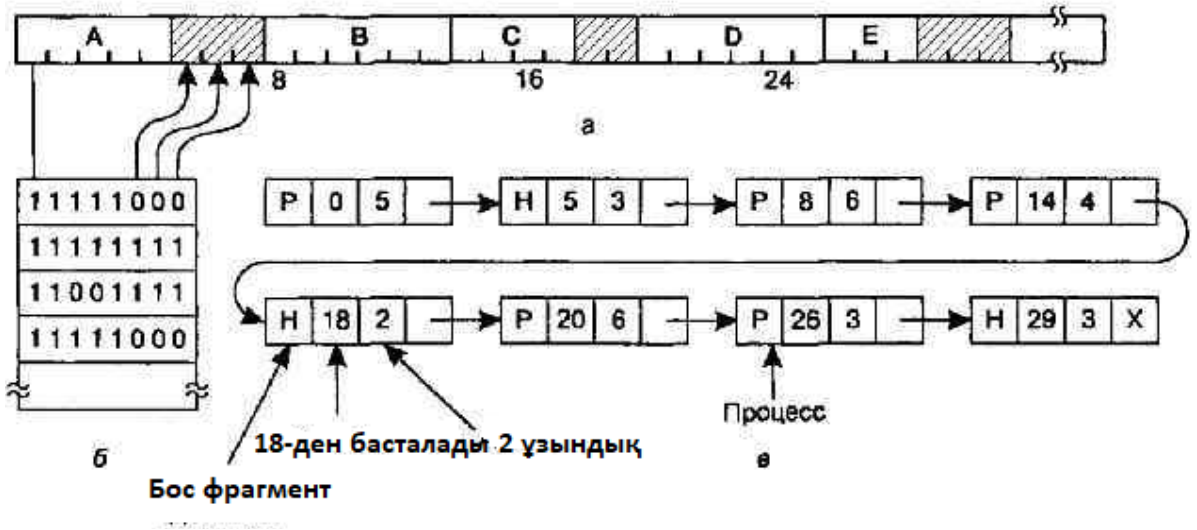
Егер компьютерде жады көлемі барлық процестерді орналастыру үшін жеткілікті болса, онда осы уақытқа дейін қарастырылған барлық схемалар белгілі бір дәрежеде жұмыс істейді. Бірақ іс жүзінде барлық процестерді орналастыру үшін қажет жедел жадтың жалпы көлемі көбінесе жедел жадының көлемінен едәуір асады. Кәдімгі Windows-, OS X - немесе Linux жүйелерінде компьютер іске қосылған кезде 50-100 немесе одан да көп процестер жұмыс істей алады. Мысалы, Windows қосымшасын орнатқан кезде, жүйе келесі іске қосылған кезде процесті бастау үшін пәрмендер жиі шығарылады, оның жалғыз міндеті осы қосымшаның жаңартуларын тексеру болады. Мұндай процесс 5-10 МБ жадыны оңай алуы мүмкін. Қалған фондық процестер кіріс поштасының, кіріс желілік қосылымдардың және тағы басқалардың болуын тексереді. Мұның бәрі алғашқы пайдаланушы бағдарламасы іске қосылмас бұрын. Photoshop сияқты заманауи қатты қолданбалы бағдарламалар 500 Мб жадыны іске қосу үшін оңай талап ете алады, ал деректерді өңдеу кезінде көптеген гигабайт алады. Демек, жадыдағы барлық процестердің тұрақты мазмұны үлкен көлемді қажет етеді және жады жетіспеушілігімен жүзеге асырыла алмайды.

Жылдар өте келе жадының шамадан тыс жүктелуін жеңу үшін екі негізгі тәсіл жасалды. Свопинг деп аталатын ең қарапайым нәрсе - бүкіл процесті жадыға орналастыру, оны біраз уақытқа іске қосу, содан кейін дискіге қалпына келтіру. Белсенді емес процестер көбінесе дискіде сақталады және жұмыс істемейтін күйде жедел жадының кеңістігін алмайды (дегенмен олардың кейбіреулері өз жұмыстарын орындау үшін мезгіл-мезгіл іске қосылады, содан кейін олар қайтадан тоқтатылады). Екінші тәсіл виртуалды жады деп аталады, ол бағдарламаларды жедел жадыда жартылай болса да іске қосуға мүмкіндік береді.

6.2 Бос жадыны басқару

6.2.1 Биттік матрицалардың көмегімен жадыны басқару

Бит матрицаларын қолданған кезде жад бірнеше сөзден бірнеше килобайтқа дейінгі бірлік блоктарға бөлінеді. Бит матрицасындағы бір бит әр блокпен байланысады, оның құрамында бірлік блок бос болса 0, ал бос болмаса 1 болады (немесе керісінше). 13-суретте жадтың бір бөлігі және оған сәйкес бит матрицасы көрсетілген.



13-сурет - Бес процестен және үш бос кеңістіктен тұратын жадының бөлігі, бір жады блоктары тік соққылармен бөлінеді: а - штрихталған аймақтар (бит матрицасында 0 сәйкес келеді) бос; б - тиісті бит матрицасы; в - тізім түріндегі бірдей ақпарат

Әзірлеуші үшін маңызды мәселе - бір жады блогының мөлшері. Блок неғұрлым аз болса, бит матрицасы соғұрлым үлкен болады. Бірақ өлшемі 4 байт болатын осындай кішкентай жалғыз жады блогымен бірге 32 биттік жады үшін 1 биттік матрица қажет болады. 32n биттен тұратын жады матрицаның N битін қолданады, сондықтан биттік матрица жадының тек 1/32 бөлігін алады. Егер үлкенірек жады блогы таңдалса, биттік матрица кішірек болады, бірақ содан кейін процестің соңғы блогында, егер ол бірлік блоктың өлшеміне дәл сәйкес келмесе, айтарлықтай жады жоғалады.

Биттік матрица жадының белгілі бір көлемінде жады сөздерін бақылаудың қарапайым әдісін ұсынады, өйткені оның мөлшері тек жадының көлеміне және жады блогының көлеміне байланысты болады. Негізгі мәселе, k бірлік блоктарын алатын процесті жадыға орналастыруды шешкен кезде, жады менеджері бит матрицасында нөлдік биттердің үздіксіз тізбегін іздеуі керек. Берілген ұзындық тізбегінің бит матрицасынан іздеу өте баяу жұмыс істейді (өйткені реттілік матрицадағы сөздердің шекараларын кесіп өтуі мүмкін) және бұл жағдай бит матрицаларын қолдануға қарсы дәлел ретінде қызмет етеді.

6.2.2 Байланысты тізімдер арқылы жадыны басқару

Жадыны бақылаудың тағы бір тәсілі - бөлінген және бос жады сегменттерінің байланысты тізімдерін жүргізу, онда сегмент процесті

қамтиды немесе екі процесс арасындағы бос орын болып табылады. 13, а-суретте көрсетілген жады бөлімі, 13, в-суретте сегменттердің байланыстырылған тізімі ретінде көрсетілген. Тізімдегі әрбір жазба белгіні сақтайды, «тесік» сегменті - hole (H) немесе процесс - процесс (P), сегмент басталатын мекенжай, оның ұзындығы және келесі жазбаның көрсеткіші болады.

Бұл мысалда сегменттер тізімі мекенжайлар бойынша сұрыпталған. Бұл сұрыптаудың артықшылығы - процесс аяқталған кезде немесе оны дискіге ауыстыру тізімді жаңартуды жеңілдетеді. Аяқталған процестің әдетте екі көршісі бар (ол жадының жоғарғы немесе төменгі адрестерінде болған жағдайларды қоспағанда). Бұл көршілер 14-суретте көрсетілген төрт комбинацияны құрайтын процестер немесе бос орындар болуы мүмкін. 14, а-суретте және тізімді жаңарту P-ны h-ке ауыстыруды қажет етеді. 14, б және в-суретте екі жазба біреуіне біріктіріліп, тізім бір жазбаға қысқарады. 14, г-суретте үш жазба біріктіріліп, тізімнен екі жазба жойылады.

Аяқталған процеске қатысты процестер кестесіндегі жазба, әдетте, осы процесс үшін тізімдегі жазбаны көрсететіндіктен, 14, в, а-суретте көрсетілгендей, бір байланыстырылған тізімді емес, екі байланыстырылған тізімді жүргізу ыңғайлы болуы мүмкін. Мұндай құрылым алдыңғы жазбаны табуды және біріктіру мүмкіндігін анықтауды жеңілдетеді.

14-сурет - Аяқталған X процесі үшін көршілердің төрт комбинациясы

6.3 Беттерді ауыстыру алгоритмдері

6.3.1 Жақында қолданылған бетті алып тастау алгоритмі

Операциялық жүйеге беттердің сұранысының пайдалы статистикасын жинауға мүмкіндік беру үшін виртуалды жадыны қолданатын компьютерлердің көпшілігінде әр бетке байланысты екі К және М күй биттері болады. R биті бетке жүгінген сайын орнатылады (оқу немесе жазу кезінде). Бит бетке жазба жасалған кезде орнатылады (яғни ол өзгертілген кезде). Бұл биттер бет кестесінің әр жазбасында бар. Бұл биттерді жадыға әр келген сайын жаңартып отыру керек, сондықтан олардың мәндерін аппараттық құралмен орнату қажет. Бит 1-ге орнатылғаннан кейін, ол операциялық жүйе қалпына келтірілгенге дейін осы мәнді сақтайды.

Егер беттің жоқ болу қатесі пайда болса, операциялық жүйе барлық беттерді қарап шығады және R және М биттерінің ағымдағы мәндеріне сүйене отырып, оларды төрт класқа бөледі:

1) 0 - класс: соңғы уақытта өтініштер де, модификациялар да болған жоқ;

2) 1 - класс: соңғы уақытта өтініштер болған жоқ, бірақ бет өзгертілген;

3) 2 - класс: соңғы уақытта айналымдар болды, бірақ модификациялар болған жоқ;

4) 3 - класс: соңғы уақытта үндеулер де, модификациялар да болды.

Бір қарағанда, 1 - класс беттері болуы мүмкін емес, бірақ егер олар 3-класс беттерінде таймердің үзілісі бойынша R биті қалпына келтірілсе, олар пайда болады. Бұл үзілістер М битін қалпына келтірмейді, өйткені ондағы ақпарат дискіде сақталған бетті қайта жазу керек пе, жоқ па білу үшін қажет. R битін М битін қалпына келтірместен қалпына келтіреді және 1-класс беттеріне әкеледі.

Жақында қолданылған бетті алып тастау алгоритмі (Recently Used (LRU)) ең төменгі бос емес сыныпқа қатысты еркін бетті жояды. Бұл алгоритмде идея бар, оның мәні, ең болмағанда, жүйелік сағаттың соңғы сағатында (әдетте бұл уақыт шамамен 20 мс), қатты пайдаланылған бетті жойғаннан гөрі, өзгертілмеген бетті жойған дұрыс. NRU алгоритмінің басты тартымдылығы - оны түсіну оңай, оны жүзеге асыру және одан өнімділікке қол жеткізу оңай, бұл, әрине, оңтайлы емес, бірақ өте қолайлы болуы мүмкін.

6.3.2 «Бірінші келді, бірінші және кетті» алгоритмі

Бетті ауыстырудың тағы бір арзан алгоритмі - FIFO алгоритмі (First In, First Out – «бірінші келді, бірінші кетті»).

Операциялық жүйе қазіргі уақытта жадыда орналасқан барлық беттердің тізімін жүргізеді, сонда жақында келгендер - тізімнің соңында, бәрінен ерте келгендер тізімнің басында орналасқан. Егер беттің жоқ болу қатесі пайда болса, тізімнің басындағы бет жойылады және оның соңына жаңа бет қосылады. Компьютерлерге қатысты мәселе туындауы мүмкін: ең ескі бет әлі де пайдалы болуы мүмкін. Сондықтан FIFO принципі оның таза түрінде сирек қолданылады.

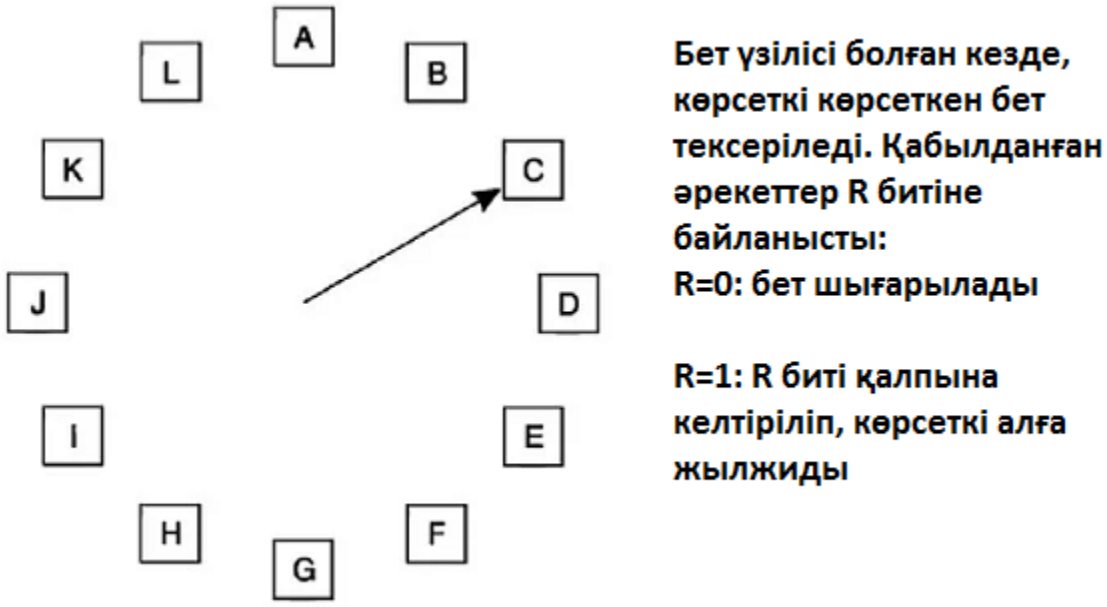
6.3.3 «Екінші мүмкіндік» алгоритмі

Жиі талап етілетін бетті жою мәселесін болдырмайтын FIFO алгоритмінің қарапайым модификациясы ең ескі беттің R битін тексеру болуы мүмкін. Егер оның мәні нөлге тең болса, онда бет ескі ғана емес, сонымен бірге, талап етілмеген, сондықтан ол дереу жойылады. Егер R биті 1-ге тең болса, ол қалпына келтіріліп, бет тізімнің соңына қойылады және оны жүктеу уақыты жадыға енгендей жаңартылады. Содан кейін іздеу жалғастырылады.

«Екінші мүмкіндік» алгоритмі бұрын жадыға жүктелген бетті іздеумен айналысады, оған өткен таймер уақыт аралығында үндеулер болмаған. Егер үндеулер барлық беттерге қатысты болса, онда «екінші мүмкіндік» алгоритмі қарапайым FIFO алгоритміне айналады. Атап айтқанда, барлық беттерде R биті орнатылған деп елестетіп көрейік. Операциялық жүйе беттерді тізімнің соңына жылжытады, әр бет тізімнің соңына қосылған сайын R битін жояды. Ақыр соңында, ол R биті қазір қалпына келтірілген бастапқы бетке оралады. Содан кейін бұл бет шығарылады. Осылайша, алгоритм әрқашан өз жұмысын аяқтайды.

6.3.4 «Сағат» алгоритмі

Барлық логикалығы бола тұра «екінші мүмкіндік» алгоритмі тым тиімсіз, өйткені ол өз тізіміндегі беттерді үнемі жылжытады. Барлық бет блоктарын сағат түрінде циклдік тізімде ұстаған дұрыс (15-сурет). Көрсеткі ең ескі бетті көрсетеді.



15-сурет – «Сағат» алгоритмі

Беттің жоқ болу қатесі орын алған жағдайда, көрсеткі көрсеткен бет тексеріледі. Егер оның R биті 0 болса, бет шығарылады, оның орнына «циферблат» ішіне жаңа бет салынып, көрсеткі бір позицияға алға жылжиды. Егер R битінің мәні 1 болса, онда ол қалпына келтіріледі және көрсеткі келесі бетке жылжиды. Бұл процесс R = 0 беті табылғанша қайталанады.

6.3.5 Ең аз талап етілетін бетті ауыстыру алгоритмі

Оңтайлы алгоритм негізінде жақсы жақындаудың негізі бірнеше соңғы командалар көп қолданатын беттерді келесі бірнеше командалар қайтадан талап етуі мүмкін екенін байқау болып табылады. Керісінше, ұзақ уақыт бойы талап етілмеген беттер ұзақ уақыт бойы талап етілмеген болып қалады. Бұл ой толығымен іске асырылатын алгоритмге итермелейді: егер беттің жоқтығында қате пайда болса, ұзақ уақыт бойы талап етілмеген беттен құтылу керек. Бұл стратегия ең аз сұранысқа ие болған бетті ауыстыру деп аталады (Least Recently Used (LRU)).

LRU алгоритмін теориялық тұрғыдан жүзеге асыру әбден мүмкін, бірақ оны практика жүзінде іске асыру оңай емес. Оны толық іске асыру үшін жадыдағы барлық беттердің байланысты тізімін жүргізу қажет. Бұл тізімнің басында жаңа ғана талап етілетін бет болуы керек, ал соңында ең аз талап етілетін бет болуы керек. Қиындығы - бұл тізім жадыға әр жүгінген сайын жаңартылуы керек. Тізімнен бетті іздеу, оны жою және содан кейін осы бетті

алға жылжыту үшін, тіпті ол аппараттық құралға жүктелген болса да, көп уақыт кетеді (егер мұндай жабдықты жасауға болады деп болжасақ).

Арнайы жабдықты қолдана отырып, LRU-ны жүзеге асырудың басқа тәсілдері бар. Оны жүзеге асыру үшін аппараттық құрал 64 биттік С санауышымен жабдықталуы керек, оның мәні әр пәрменнен кейін автоматты түрде артады. Сонымен қатар, беттер кестесіндегі әр жазбада осы санауыштың мәні болуы үшін жеткілікті түрде үлкен өріс болуы керек. Жадыға әр хабарланғаннан кейін С санауышының ағымдағы мәні осы үндеу болған бетке қатысты беттер кестесінің жазбасында сақталады. Егер беттің жоқ болу қатесі пайда болса, операциялық жүйе ең кішісін табу үшін беттер кестесіндегі барлық санауыштарды тексереді. Бұл мән қай беттің жазбасына қатысты болса, сол бет ең аз сұранысқа ие болады.

6.4 Бет өлшемі

Бет өлшемі - бұл операциялық жүйе таңдайтын параметр. Егер жабдық, мысалы, 4096 байттағы беттер үшін жасалған болса да, операциялық жүйе 0 және 1, 2 және 3, 4 және 5 беттерінің жұптарын 8 Кбайт өлшемді беттер ретінде оңай қарастыра алады, әрқашан оларға 8192 байт өлшемді екі дәйекті бет блоктарын бөліп көрсетеді.

Беттің ең жақсы өлшемін анықтау бірнеше бәсекелес факторлар арасындағы тепе-теңдікті сақтауды талап етеді. Осылайша, абсолютті оңтайлы шешім болмайды. Ең алдымен беттің кішкене өлшеміне дәлел болатын екі факторды алайық. Кездейсоқ алынған мәтін, деректер немесе стек сегменті беттердің бүтін санын толтырмайды. Орташа алғанда, соңғы беттің жартысы бос қалады. Бұл беттегі қалған бос орын ысырап болады. Бұл шығындар ішкі фрагментация деп аталады. Егер N сегменттерінің жадысында және беттің өлшемі p байт болса, онда NP/2 байт ішкі фрагментацияға жұмсалады. Бұл ойлар беттің кішігірім өлшеміне дәлел болып табылады.

Беттің кішігірім өлшеміне қатысты тағы бір дәлел сегіз кезеңнен тұратын бағдарламаны қарастырған кезде пайда болады, олардың әрқайсысы 4 Кбайт. Бет өлшемі 32 Кбайт болған кезде, бағдарлама әрқашан 32 Кбайтты бөлуі керек. 16 Кбайт өлшемді бетте оған тек 16 Кбайт қажет болады. Беттің өлшемі 4 Кбайт немесе одан аз болса, кез-келген уақытта оған тек 4 Кбайт қажет болады. Жалпы, беттің үлкен мөлшерімен жадыда кішкентайға қарағанда пайдаланылмаған кеңістік көп болады.

Сонымен қатар, көлемі аз беттерде бағдарламаларға саны көп беттер қажет болады, бұл беттер кестесінің үлкен көлеміне әкеледі. 32 Кбайт бағдарламасы тек 8 Кб үшін 4 бетті, бірақ 512 Байттан 64 бетті қажет етеді. Әдетте, бүкіл бет бірден дискіге және дискіден жіберіледі, ал уақыттың көп

бөлігі қажетті секторды табуға және дискіні жылжытуға байланысты кідірістерге жұмсалады, сондықтан кішкене бетті беру үлкен бетті тасымалдаумен бірдей уақытты алады. 512 Байттан 64 бетті жүктеу үшін 64 • 10 мс қажет болуы мүмкін, ал 8 Кбайттан төрт бетті жүктеу үшін тек 4 • 12 мс қажет.

Сонымен қатар, кішігірім беттер TLB-де көптеген құнды кеңістікті пайдаланады. Сіздің бағдарламаңызда 64 Кбайт жұмыс жиынтығы бар 1 МБ жады қолданылады делік. 4 Кбайт өлшемді беттерде бағдарлама TLB-де кем дегенде 16 жазбаны алады. 2 Мб өлшемді беттерде TLB-де бір жазба жеткілікті болады (теориялық тұрғыдан сізге деректер мен нұсқауларды бөлу қажет болуы мүмкін). TLB жазбаларының жетіспеушілігіне және олардың өнімділікке айтарлықтай әсер етуіне байланысты үлкен беттерді пайдалану арқылы төлеуге болады. Осы сауда-саттықтың барлығын теңестіру үшін операциялық жүйелер кейде жүйенің әртүрлі бөліктері үшін әртүрлі бет өлшемдерін пайдаланады. Мысалы, ядро үшін үлкен беттер және жеке процестер үшін кішірек беттер.

Кейбір машиналарда орталық процессорды бір процестен екіншісіне ауыстырған сайын беттер кестесін (операциялық жүйе арқылы) аппараттық регистрлерге жүктеу керек. Бұл машиналар үшін кішігірім беттердің болуы беттің көлемін азайту кезінде бет регистрлерін жүктеуге кететін уақытты көбейтуді білдіреді. Сонымен қатар, беттің өлшемі азайған сайын, беттер кестесінің кеңістігі артады.

Соңғы тұжырым математикалық тұрғыдан талдануы мүмкін. Процестің орташа өлшемі s байт, ал бет өлшемі p байт болсын. Сонымен қатар, әр бетке жазу үшін e байт қажет делік. Содан кейін әр процеске қажет беттердің шамамен саны S/p болады, ол Se/p байт беттер кестесінің кеңістігін алады. Ішкі фрагментацияға байланысты процестің соңғы бетіндегі пайдаланылмаған жады кеңістігі p/2 болады. Осылайша, беттер кестесі мен ішкі фрагментацияның жалпы шығындары осы екі шаманы қосу арқылы алынады:

Шығындар = se / p + p / 2.

Бірінші термин (беттер кестесінің өлшемі) беттің кішкентай өлшемімен үлкен мәнге ие болады. Екінші қосқыш (ішкі фрагментация) беттің үлкен өлшемімен үлкен мәнге ие болады. Ең жақсы нұсқа ортасында орналасқан. Егер сіз p айнымалысының бірінші туындысын алып, оны нөлге теңестірсеңіз, онда біз мына теңдеуді аламыз:

–se/p2 + 1/2 = 0.

Осы теңдеуден беттің оңтайлы өлшемін беретін формуланы алуға болады (фрагментация үшін жадының жоғалуын және беттер кестесінің өлшемін ескере отырып). Нәтиже келесідей болады:

$p=\sqrt{2se}$.

S = 1 Мб және e = 8 байт үшін беттер кестесіндегі әр жазба үшін оңтайлы бет өлшемі 4 Кбайт болады. Қолжетімді компьютерлер 512 байттан 64 Кб-қа дейінгі бет өлшемін пайдаланады. Бұрын 1 Кбайт өлшемі жиі қолданылған, бірақ қазір 4 Кбайт бетінің өлшемі жиі кездеседі.

6.5 Ортақ беттер

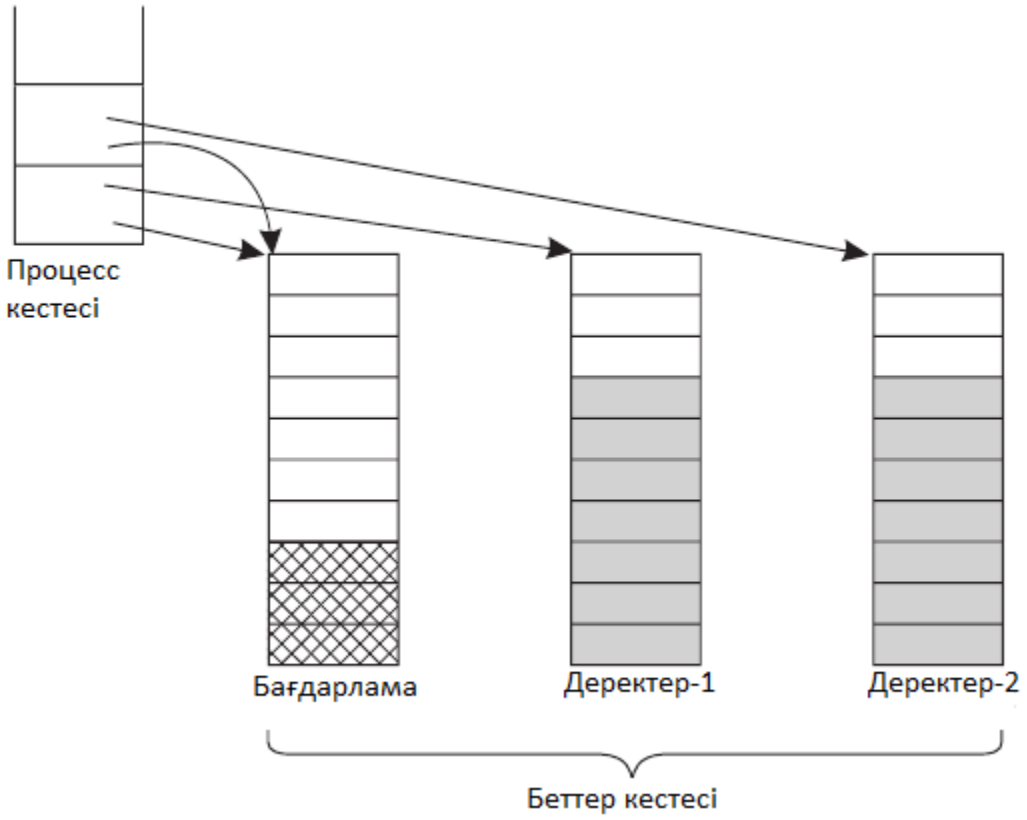
Әзірлеудің тағы бір мәселесі – ресурстарды ортақ пайдалану. Үлкен көп функциялы жүйелерде бір бағдарламаны бірнеше қолданушы бір уақытта қолдануы жиі кездеседі. Тіпті жеке пайдаланушы бір кітапхананы қолдана отырып, бірнеше бағдарламаны іске қоса алады. Сонымен қатар, бір парақтың екі данасының жадысында бір уақытта болуын болдырмау үшін беттерді бөлісудің тиімділігі айқын. Мәселе мынада, барлық беттерді ортақ пайдалануға болмайды. Атап айтқанда, тек оқуға арналған беттерді, мысалы, мәтін бағдарламалары бар беттерді бірге ортақ пайдалануға болады, ал деректер беттерін бөлісу қосымша қиындықтармен байланысты.

Егер жеке I-және D-кеңістіктерге қолдау көрсетілсе, I-кеңістіктері үшін бірдей беттер кестесін қолданатын бір немесе бірнеше процестердің болуына байланысты бағдарламаларды ортақ пайдалану міндеті салыстырмалы түрде қарапайым болады, бірақ D-кеңістіктері үшін әртүрлі беттер кестелері. Әдетте, беттерді бөлісуді қолдайтын енгізулерде беттер кестесі - бұл процесс кестесінен тәуелсіз мәліметтер құрылымы. Сонымен қатар, 16-суретте көрсетілгендей, өзінің процесс кестесіндегі әр процестің екі көрсеткіші бар: біреуі I-кеңістіктің беттер кестесінде, екіншісі D-кеңістіктің беттер кестесінде. Процесс жоспарлаушысы іске қосылатын процесті таңдаған кезде, ол осы көрсеткіштерді беттер кестелерінің орналасқан жерін және оларды жады менеджерімен (MMU) орнату үшін пайдаланады. Процестер бағдарламаларды (немесе кейде кітапханаларды) жеке I-және D-кеңістіктер болмаған кезде де бөлісе алады, бірақ бұл үшін күрделі механизм қолданылады.

Екі немесе одан да көп процесс бірдей кодты бөліскенде, ортақ беттерде проблема туындайды. Екі процесс, A және B, редакторды іске қосады және оның беттерін бөліседі делік. Егер жоспарлаушы а процесін жадыдан алып тастау туралы шешім қабылдаса, оның барлық беттерін құрбан етіп, бос бет блоктарын басқа бағдарламамен толтырса, бұл B процесінің көптеген беттің жоқ болу қателіктерін тудырады және сол беттерді қайтарады.

Сондай-ақ, A процесін тоқтатқан кезде, олардың дискілік кеңістігі кездейсоқ босатылмауы үшін қай беттер әлі де қолданылатынын анықтау керек. Беттің ортақ пайдалануын анықтау үшін барлық бет кестелерін қарау

әдетте өте қымбат операция болып табылады, сондықтан ортақ беттерді бақылау үшін арнайы деректер құрылымы қажет болады, әсіресе егер ортақ пайдалану тақырыбы бүкіл беттер кестесі емес, бөлек бет (немесе бірнеше бет) болса.



16-сурет - Бір бағдарламаны бөлісетін екі процестің ортақ беттер кестелері бар

Деректерді бөлісу кодты бөлісуден гөрі қиын, бірақ бұны мүмкін емес деп айтуға болмайды. Атап айтқанда, Unix-те fork жүйелік шақыруынан кейін ата-ана мен еншілес процестері бағдарлама мәтінін де, деректерді де бөлісуге мәжбүр. Жадыны бет ұйымы бар жүйелерде осы процестердің әрқайсысына жеке беттер кестесін ұсыну және олардың екеуінде де бірдей беттер жиынтығына сілтегіш болу жиі қолданылады. Осылайша, fork жүйелік шақыруын орындау кезінде беттерді көшіру болмайды. Дегенмен, барлық деректер беттері әр процесс үшін тек оқуға арналған беттер ретінде көрсетіледі - READ ONLY.

Екі процесс тек деректерді өзгертпестен оқып жатқанда, бұл жағдай жалғасуы мүмкін. Кез-келген процесс жады сөзін жаңартқаннан кейін, тек оқу режимін қорғауды бұзу операциялық жүйенің жүйелік үзілуіне әкеледі. Осыдан кейін осы жағдайды тудырған беттің көшірмесі жасалады, сондықтан

әр процестің өз данасы болады. Енді екі көшірме үшін де оқу-жазу режимі - READ-WRITE орнатылады, сондықтан әр көшірмедегі келесі жазбалар жүйелік үзілістерсіз орын алады. Бұл стратегия ешқашан өзгермейтін беттер (бағдарламаның барлық беттерін қоса) көшірудің қажеті жоқ дегенді білдіреді. Көшіру тек өзгеріске ұшыраған беттер үшін қажет. Жазу кезінде көшіру (жазу кезінде көшіру) деп аталатын бұл тәсіл көшіру көлемін азайту арқылы өнімділікті арттырады.

6.6 Ортақ кітапханалар

Ортақ пайдалану басқа құрылымдық деңгеймен болуы мүмкін. Егер бағдарлама екі рет іске қосылса, онда көптеген операциялық жүйелер автоматты түрде мәтіндік беттерді бөлісуді ұйымдастырады, сондықтан жадыда тек бір көшірме болады. Мәтіндік беттер әрқашан тек оқу режимінде қолданылады, сондықтан ешқандай проблемалар туындамайды. Операциялық жүйеге байланысты әр процесс деректері бар беттердің жеке көшірмесін ала алады немесе оларды бөлісуге және «тек оқу үшін» деп белгілеуге болады. Егер қандай да бір процесс деректер бетін өзгертсе, ол үшін жеке көшірме жасалады, яғни жазуға жарамды көшірме қолданылады.

Қазіргі жүйелерде көптеген процестер қолданатын көптеген үлкен кітапханалар бар, мысалы, көптеген енгізу-шығару кітапханалары және графикалық кітапханалар. Осы кітапханалардың барлығын дискідегі әр орындалатын бағдарламаға статикалық байланыстыру оларды іс жүзінде болғаннан гөрі одан да көп етеді.

Оның орнына Windows-та динамикалық қосылатын кітапханалар (Dynamic Link Libraries (DLL)) деп аталатын ортақ пайдаланылатын (ортақ) кітапханалардың жалпы технологиясы қолданылуы керек. Ортақ кітапхана идеясын түсінікті ету үшін алдымен дәстүрлі реттеуді қарастырыңыз. Бағдарламаны командада реттеу кезінде сілтеме жасаушыға бір немесе бірнеше объектілік файлдар және мүмкін бірнеше кітапханалар, мысалы, UNIX командасында көрсетіледі:

ld *.o –lc –lm.

Ағымдағы каталогтағы .o (object) кеңейтілімі бар барлық файлдарды құрастырады, содан кейін екі кітапхананы /usr/lib/libc сканерлейді а және / usr/lib / libm.a. объект файлдарында шақырылған, бірақ оларда жоқ кез-келген функциялар (мысалы, printf) белгісіз сыртқы функциялар деп аталады және кітапханаларда ізделеді. Егер олар табылса, олар орындалатын екілік файлға қосылады. Кез-келген шақырылған, бірақ жоқ функциялар да белгісіз сыртқы функцияларға айналады. Мысалы, printf функциясы жазу функциясын қажет етеді, сондықтан егер жазу функциясы әлі қосылмаған болса, құрастырушы

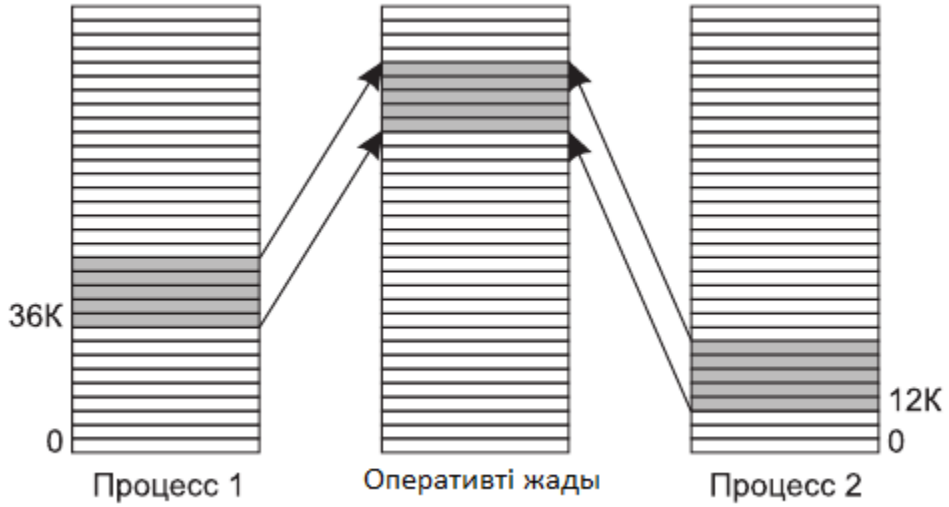
оны іздейді және оны тапқаннан кейін екілік файлға қосады. Орналасу аяқталғаннан кейін дискіге жазылған орындалатын екілік барлық қажетті функцияларды қамтиды. Кітапханада бар, бірақ талап етілмеген функциялар оған кірмейді. Бағдарлама жадыға жүктелгенде және орындалғанда, ол барлық қажетті функцияларды қамтиды.

Енді кәдімгі бағдарламада графика мен пайдаланушы интерфейсіне қатысты 20-50 МБ мүмкіндіктер қолданылады делік. Барлық осы кітапханалары бар статикалық түрде жинақталған жүздеген бағдарламалар үлкен дискілік кеңістікті, сондай-ақ олар жүктелгеннен кейін жедел жады кеңістігін ысырап етеді, өйткені жүйеде бұл кітапханаларды ортақ ретінде пайдалануға болатындығы туралы білуге мүмкіндік болмайды. Содан кейін ортақ кітапханалар сахнаға шығады. Бағдарлама кітапханаларды бөлісуді ескере отырып құрастырылған кезде (бұл статикалық орналасудан біршама ерекшеленеді), нақты шақырылған функцияларды қосудың орнына, құрастырушы орындау барысында шақырылған функцияға қосылатын шағын бітеуіш бағдарламаны қамтиды. Жүйеге немесе конфигурация ерекшеліктеріне байланысты ортақ кітапханалар бағдарламаны жүктеу кезінде немесе олардағы функциялар бірінші рет шақырылған кезде жүктеледі. Әрине, егер ортақ кітапхана басқа бағдарламамен жүктелген болса, онда оны қайта жүктеудің қажеті жоқ - бар мән-мағына осында. Айта кету керек, ортақ кітапхананы жүктеу немесе пайдалану кезінде бүкіл кітапхана бірден жадыда оқылмайды. Ол қажет болған жағдайда беттен кейін жүктеледі, сондықтан шақырылмаған функциялар жедел жадыға берілмейді.

Ортақ кітапханалардың тағы бір артықшылығы - орындалатын файлдарды кішірейтуге және жады кеңістігін үнемдеуге көмектеседі: егер ортақ кітапханадағы функция қатені жою үшін жаңартылса, оны тудыратын бағдарламаны қайта құрастырудың қажеті жоқ. Ескі екілік файлдар жұмыс қабілеттілігін сақтайды. Бұл қасиет коммерциялық бағдарламалық қамтамасыз ету үшін ерекше мәнге ие болады, оның коды клиентке жеткізілмейді. Мысалы, егер Microsoft белгілі бір стандартты DLL кітапханасында жүйенің қауіпсіздігіне әсер ететін қатені тауып, түзетсе, жаңарту бағдарламасы - Windows Update - жаңа DLL жүктеп, оны ескі кітапханамен алмастырады және осы DLL қолданатын барлық бағдарламалар келесі іске қосу кезінде автоматты түрде жаңа нұсқаны пайдаланады.

Бірақ ортақ кітапханалар бізге шешуді қажет ететін бір кішкентай проблемамен келді. Бұл мәселе 17-суретте көрсетілген. Мұнда 20 Кбайт кітапхананы бөлісетін екі процесс көрсетілген (оның әр блогы 4 Кбайт алады делік). Бірақ әр процестегі кітапхана әртүрлі мекенжайларда орналасқан, өйткені бағдарламалардың өздері бірдей емес. 1-процесте кітапхана 36 К мекенжайынан басталады; 2-процесте оны орналастыру 12 К мекенжайынан

басталады. Кітапхананың бірінші функциясы бірінші кезекте кітапханадан 16 мекенжайға өту керек делік. Егер кітапхана бірлесіп пайдаланылмаса, оны жүктеу кезінде жылдам ауыстыруға болады, сондықтан (1-процесте) 36-дан + 16-ға дейін виртуалды мекенжайға ауысуға болады. Айта кету керек, кітапхана орналасқан жедел жадының физикалық мекенжайы барлық беттер виртуалды адрестерден физикалық адрестерге жады менеджерінің жабдықтары - MMU көрсетілгенше маңызды емес.



17-сурет - Екі процесте қолданылатын ортақ кітапхана

Бірақ кітапхана бірге қолданыла бастағанда, көшіп орын ауыстыру енді жұмыс істемейді. Ақыр соңында, бірінші функция 2-процеспен шақырылған кезде (12 К мекенжайы бойынша), ауысу командасы оны 36 К + 16 мекенжайына емес, 12 К + 16 мекенжайына жіберуге мәжбүр болады. Бұл шағын мәселе. Оны шешудің бір жолы - ортақ кітапхананы пайдаланатын әр процесс үшін жаңа беттерді жазу және құру кезінде көшірмені пайдалану және оларды жасау кезінде жылдам қозғалу. Бірақ бұл схема, әрине, кітапхананы бөлісудің барлық мақсатына нұқсан келтіреді.

Ең жақсы шешім - ортақ кітапханаларды компиляторға арналған арнайы құсбелгімен құрастыру, бұл компиляторға абсолютті адресті қолданатын командалар жасамауға нұсқау береді. Оның орнына салыстырмалы адресті қолданатын командалар ғана қолданылады. Мысалы, әрдайым дерлік N байтқа алға (немесе артқа) ауысуды бұйыратын команда бар (ауысу үшін нақты мекен-жай беретін командаға балама ретінде). Бұл команда ортақ кітапхананы виртуалды мекен-жай кеңістігіне орналастыруға қарамастан дұрыс жұмыс істейді. Мәселе абсолютті адресті алып тастау арқылы шешілуі мүмкін. Тек салыстырмалы ығысуларды қолданатын код позициялық тәуелсіз код деп аталады.

7 ТЕЛЕКОММУНИКАЦИЯЛЫҚ ҚОЛЖЕТІМДІЛІКТІ БАСҚАРУ

7.1 Құжат негізіндегі байланыстырушы бағдарламалық қамтамасыз ету

Бүкіләлемдік ғаламтор өте қарапайым бастапқы парадигмаға негізделген: әр компьютерде веб-парақтар деп аталатын бір немесе бірнеше құжаттар болуы мүмкін. Әр веб-парақта мәтін, суреттер, белгішелер, дыбыстар, бейнеклиптер және т.б., сондай-ақ, басқа веб-парақтарға сілтемелер (сілтемелер) бар. Пайдаланушы веб-парақты сұраған кезде парақты экранда көрсететін веб-шолғыш деп аталатын бағдарлама қолданылады. Сілтемені нұқу экрандағы ағымдағы бетті сілтеме көрсетілген бетке ауыстырады. Соңғы уақытта Бүкіләлемдік ғаламторға әртүрлі әшекейлер әкелінгенімен, оның негізінде парадигма әлі де күшінде: бұл басқа құжаттарды көрсете алатын құжаттардың үлкен, бағытталған графы.

Әр веб-парақта URL деп аталатын ерекше мекенжай бар (Uniform Resource Locator-бірыңғай ресурс көрсеткіші), онда хаттама атауы, қос нүкте, қос сызық, DNS атауы, қиғаш сызық және файл атауы: protocol://DNS-аты/файл атауы. Көбінесе http хаттама ретінде қолданылады (HyperText Transfer Protocol-гипермәтіндік файлдарды беру протоколы), бірақ сонымен бірге ftp хаттамасы және басқалары бар, содан кейін файл сақталатын хосттың DNS атауы. Сонымен, жергілікті файлдың аты қай файл қажет екенін айтады. Осылайша, URL мекен-жайы бүкіл әлем кеңістігінде нақты файлды анықтайды.

Жүйе келесідей біртұтас болып қалыптасады. Өзінің негізінде Бүкіләлемдік тор клиент - серверлік жүйе болып табылады, мұнда пайдаланушы клиент ретінде, ал веб-сайт сервер ретінде әрекет етеді. Пайдаланушы браузерге URL өрісіне теру арқылы немесе ағымдағы бетте орналасқан еренсілтемені нұқу арқылы URL мекенжайын берген кезде, шолғыш сұралған веб-бетті алу үшін белгілі бір қадамдар жасайды. Қарапайым мысал ретінде оған URL мекенжайы берілген делік http://www.minix3.org/getting-started/index.html. Содан кейін мыналар болады:

1) Браузер DNS-тен аты-жөніне сәйкес IP мекенжайын сұрайды www.minix3.org;

2) DNS жауап ретінде 66.147.238.215 береді;

3) Браузер 66.147.238.215 IP мекенжайы бар хостта 80 портына TCP қосылымын орнатады;

4) содан кейін ол getting-started/index файлына сұрау жібереді.html;

5) Сервер www.minix3.org getting-started/index файлын жібереді.html;

6) Браузер бүкіл мәтінді getting-started/index файлынан көрсетеді.html;

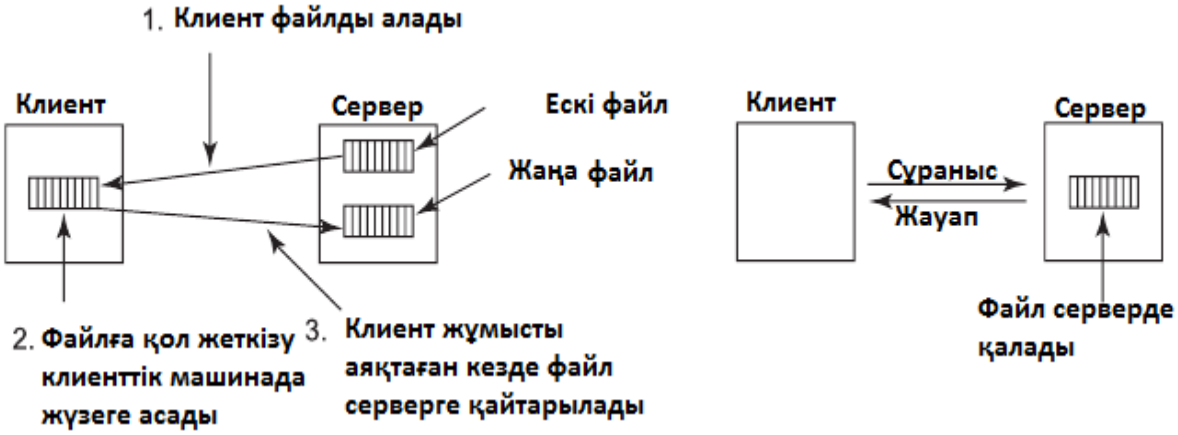
7) сонымен бірге браузер парақтағы барлық суреттерді шығарады және көрсетеді;

8) TCP қосылымы үзіледі.

7.2 Файлдық жүйеге негізделген байланыстырушы бағдарламалық қамтамасыз ету

7.2.1 Деректерді беру моделі

Бірінші сұрақ жүктеу - жүктеу моделі (upload/download model) мен қашықтан қол жеткізу моделі (remote access model) арасындағы таңдауды қарастырады. Олардың біріншісін қолданған кезде (18, а-сурет) процесс файлға жүгінеді, алдымен оны орналасқан қашықтағы серверден көшіреді. Егер файл тек оқуға арналған болса, онда жоғары өнімділікті қамтамасыз ету үшін ол жергілікті түрде оқылады. Егер файл жазылуы керек болса, онда ол жергілікті түрде жазылады. Процесс файлмен аяқталған кезде, жаңартылған файл серверге қайтарылады. Қашықтан қол жеткізу моделін қолданған кезде файл серверде қалады, ал клиент оған қайда және не істеу керектігін жібереді (18, б-сурет).



18-сурет - Модель: а - жүктеу; б - қашықтықтан қол жеткізу

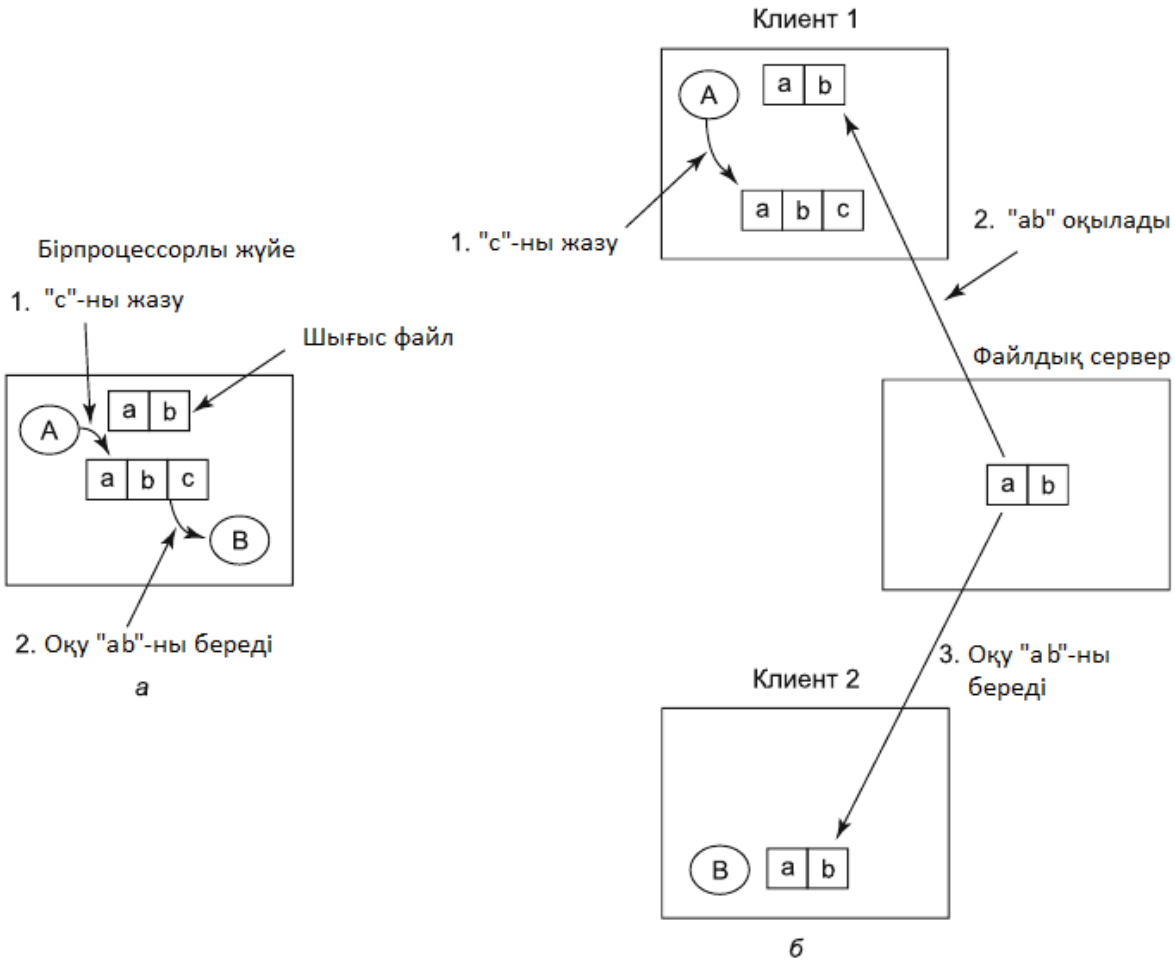
Жүктеу моделінің артықшылығы - оның қарапайымдылығы және файлды тасымалдау оны бөліктерге бөлуден гөрі тиімдірек. Бұл модельдің кемшілігі - бүкіл файлды жергілікті сақтау үшін жеткілікті орын қажет, тек оның бір бөлігі қажет болған кезде бүкіл файлды жылжытудың қисынсыздығы, сонымен қатар бірнеше қолданушы файлды параллель пайдалану кезінде туындайтын қарама-қайшылық мәселелері.

7.2.2 Файлдарды ортақ пайдалану семантикасы

Екі немесе одан да көп пайдаланушы бір файлмен бірге жұмыс істеген кезде, проблемаларды болдырмас үшін файлды оқу және жазу кезінде қолданылатын семантиканы дәл анықтау керек. Бір процессорлық жүйелерде семантикалық ережелер read жүйелік шақырудың артынан read жүйелік шақыруын орындау read жаңа жазылған мәнді қайтарады (19, а-сурет). Осыған ұқсас, егер read жүйелік шақыруы екі қатарлы write жүйелік шақыруларынан кейін бірден орындалса, онда соңғы write жүйелік шақыруларында жазылған мән оқылады. Осылайша, барлық жүйелік шақырулар жүйемен бір ретпен реттеледі және барлық процессорлар бірдей тәртіпті көреді. Біз дәйекті қарама-қайшылықсыз (sequential consistency) сияқты модельге жүгінеміз.

Таратылған жүйеде бір файлдық сервер болған кезде және клиенттерде файлдарды кэштеу болмаған кезде дәйекті қарама-қайшылықсыздыққа оңай қол жеткізіледі. Барлық read және write жүйелік шақырулары оларды қатаң ретпен өңдейтін файлдық серверге тікелей келеді.

Алайда, іс жүзінде барлық файлдық сұраулар бір серверде орындалуы керекті таратылған жүйенің өнімділігі көбінесе өте төмен көрсеткіштерге ие. Көбінесе бұл мәселе клиенттерге өздерінің кэш-жадысындағы ең көп қолданылатын файлдардың жергілікті көшірмелерімен жұмыс істеуге рұқсат беру арқылы шешіледі. Бірақ егер 1 клиенті кэште сақталған файлға жергілікті өзгерістер енгізсе және көп ұзамай 2 клиенті файлды серверден оқыса, онда екінші клиент файлдың ескірген нұсқасын алады (19, б-сурет).



19-сурет - Таратылған жүйеде файлға қол жеткізу нұсқалары:
а - дәйекті қарама-қайшылықсыздық; б - кэштелген файлды оқыған кезде ескірген мән қайтарылуы мүмкін

Бұл қиындықтарды айналып өтудің бір әдісі - кэштелген файлдардағы барлық өзгерістерді дереу серверге қайтару. Дизайнның қарапайымдылығымен бұл тәсіл тиімсіз. Балама шешім - файлдарды ортақ пайдалану семантикасын жұмсарту. Read-қа қойылатын талаптың орнына барлық алдыңғы write шақыруларының әсерін көру үшін жаңа ережені енгізуге болады: «ашық файлға енгізілген өзгерістер алдымен оларды шығаратын процесте ғана көрінеді. Басқа процестер бұл өзгерістерді файл жабылғаннан кейін ғана көреді». Осы ережені ұстану не болғанын 19, в-суретте болғанды өзгертпейді, бірақ қазір процестердің әрекеті (в процесі файлдың бастапқы нұсқасын алады) заңдастырылады. Клиент 1 файлды жапқан кезде, ол серверге файлдың өзгертілген көшірмесін жібереді, сондықтан read жүйелік шақырулары қажет болған жағдайда файлдың жаңа нұсқаларын алады. Шын мәнінде, бұл 18-суретте көрсетілген жүктеу-моделі.

Бұл семантикалық ереже кеңінен қолданылды және сеанс семантикасы (session semantics) ретінде белгілі.

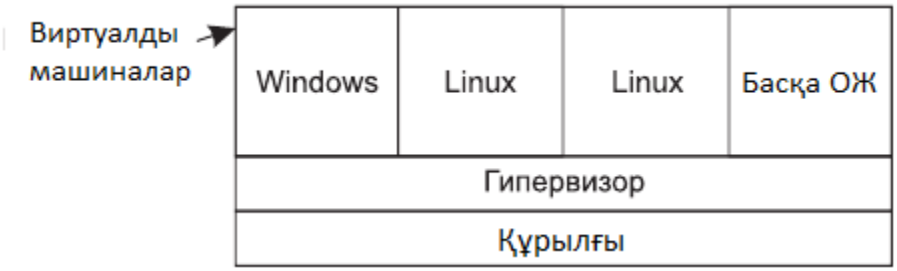
Сеанстық семантиканы қолданған кезде екі немесе одан да көп клиент бір уақытта кэште оқып, сол файлды өзгертсе не болады деген сұрақ туындайды. Шешімдердің бірі - барлық файлдар кезекпен жабылып, олардың нұсқалары серверге жіберілгендіктен, түпкілікті нәтиже соңғы нұсқалардың қайсысы жабылатынына байланысты болады. Аз жағымды, бірақ сәл қарапайым балама түпкілікті нәтиже нұсқалардың бірі болатынына келіседі, бірақ нақты таңдауды қарастырмайды.

Сеанстық семантикаға балама тәсіл - жүктеу моделін қолдану, бірақ жүктелген файлды автоматты түрде блоктау. Басқа клиенттердің файлды жүктеу әрекеттері бірінші клиент бұл файлды қайтарғанға дейін тоқтатылады. Егер файл үлкен сұранысқа ие болса, сервер файлды ұстап тұрған клиентке асығуды сұрап хабарлама жібере алады, дегенмен бұл сұраныс ештеңеге әкелмеуі мүмкін. Ең соңында, файлдарды ортақ пайдалану семантикасын дұрыс құру оңай емес, талғампаз және тиімді шешімдері жоқ.

ҚОРЫТЫНДЫ

Виртуализация және бұлт

Жабдықпен тікелей әрекеттесетін гипервизорлар (20-сурет) көптеген компьютерлерде жұмыс істейді. Гипервизор өзінің операциялық жүйелері бар көптеген виртуалды машиналарды жасайды. Бұл құбылыстың болашағы жақсы. Қазіргі уақытта көптеген компаниялар бұл идеяны басқа ресурстарды виртуализациялау арқылы алады. Мысалы, желілік жабдықты басқаруды виртуализациялауға үлкен қызығушылық бар. Бұл қызығушылық одан әрі жалғасуда — бұлтты желіні басқаруды бастау. Сонымен қатар, жеткізушілер мен зерттеушілер гипервизорларды бірқатар көрсеткіштер бойынша жетілдіріп, оларды кішірейтіп, тезірек жасайды, сонымен қатар олардың кепілдік берілген оқшаулау қасиеттерін арттырады.



20-сурет-Төрт виртуалды машинамен жұмыс жасайтын гипервизор

Үлкен мекенжай кеңістігі бар операциялық жүйелер

32 биттік машиналар 64 биттік машиналармен алмастырылғандықтан, операциялық жүйелер құрылымында басты өзгеріс болуы мүмкін. 32 биттік мекенжай кеңістігі іс жүзінде онша үлкен емес. Егер сіз 232 байтты жердің барлық тұрғындарына бөлуге тырыссаңыз, онда әрқайсысы бір байттан аз алады. Сонымен қатар, 264 шамамен 2 • 1019-ға тең. Сонымен қатар, 64 биттік мекенжай кеңістігінде планетаның әр тұрғынына 3 Гб фрагментті бөлуге болады.

2 * 1019 байттағы мекен-жай кеңістігімен не істеуге болады? Бастау үшін біз файлдық жүйе тұжырымдамасынан бас тарта аламыз. Оның орнына барлық файлдарды үнемі жадыда сақтауға болады (виртуалды). Өйткені, 4 Гб-қа дейін сығылған миллиардтан астам толық метражды фильмдер үшін жеткілікті орын бар.

Тағы бір мүмкіндік - тұрақты нысандарды пайдалану. Нысандар мекенжай кеңістігінде құрылуы мүмкін және объектіге барлық сілтемелер

жойылғанша сақталады, содан кейін объектінің өзі автоматты түрде жойылады. Мұндай нысандар компьютерді өшіріп, қайта іске қосқаннан кейін де мекенжай кеңістігінде сақталады. Барлық 64 биттік мекенжай кеңістігін толтыру үшін 5000 жыл ішінде 100 Мб/с жылдамдықпен нысандар құру керек. Әрине, көптеген деректерді сақтау үшін көптеген дискілер қажет болады, бірақ тарихта алғаш рет мекенжайлық кеңістік емес, дискілердің физикалық мүмкіндіктері шектеуші факторға айналды.

Мекенжай кеңістігінде көптеген нысандар болған кезде, нысандарды бөлісуді жеңілдету үшін бірнеше процестердің бір мекенжай кеңістігінде бір уақытта жұмыс істеуіне мүмкіндік беру қызықты болады. Мұндай схеманы қолдану, әрине, қазіргі кездегіден мүлдем өзгеше операциялық жүйелердің пайда болуына әкеледі. 64 биттік адрестерді енгізу кезінде қайта қарауға қажетті тағы бір жүйелік аспект - виртуалды жады. 264 байт виртуалды мекенжай кеңістігі мен сегіз байт беттерінде бізде 251 бет болады. Осы мөлшердегі қарапайым беттер кестелерімен жұмыс істеу оңай болмайды, сондықтан сізге басқа шешім қажет болады. Қалай болғанда да, 64 биттік операциялық жүйелердің пайда болуы жаңа үлкен зерттеу саласын жасайды.

Деректерге кедергісіз қол жеткізу

Бұл күндері пайдаланушылар кез-келген жерден және кез-келген уақытта көптеген деректерге қол жеткізе алады деп күтеді. Бұл әдетте DropBox, GoogleDrive, iCloud және SkyDrive сияқты деректерді сақтау қызметтерін қолдана отырып, бұлтты құрылымдарда деректерді сақтау арқылы жүзеге асырылады. Онда сақталған барлық файлдар желілік қосылымы бар кез келген құрылғыдан қол жетімді болуы мүмкін. Сонымен қатар, деректерге қол жеткізуге арналған бағдарламалар көбінесе бұлтты қоймада болады, сондықтан оларды кез келген жерде орнатудың қажеті жоқ. Бұл адамдарға смартфонды кез келген, тіпті ерекше жерде қолдана отырып, мәтіндік процессордың, электрондық кестелер мен презентациялардың файлдарын оқуға және өзгертуге мүмкіндік береді. Мұның бәрі әдетте прогресс үшін беріледі.

Абсолютті кедергіге қол жеткізу үшін көптеген жасырын жүйелік шешімдер қажет. Мысалы, желілік байланыс болмаса не істеу керек? Әрине, адамдардың жұмысы үзілгенін қаламас едік. Сондай-ақ, жергілікті буферге өзгертулер енгізіп, қосылымды қалпына келтіру кезінде негізгі құжатты жаңартуға болатындығы түсінікті, бірақ егер көптеген құрылғыларда бір-біріне қайшы келетін өзгерістер болса ше? Деректерді ортақ пайдалану кезінде бұл мәселе жиі кездеседі, бірақ ол жалғыз пайдаланушымен де көрінуі мүмкін. Сонымен қатар, егер файл үлкен болса, оған қол жеткізу үшін ұзақ

уақыт күткіміз келмейді. Мұндағы негізгі сұрақтар - кэштеу, алдын ала жүктеу және синхрондау. Қазіргі операциялық жүйелерде бірнеше машиналарды біріктіру бойынша жұмыс әлі де бірқатар маңызды кедергілерге ие. Әрине, біз осы саладағы жағдайды түзете аламыз.

Дербес қуатты компьютерлер

64 биттік адрестік кеңістігі бар қуатты дербес компьютерлер, жоғары жылдамдықты желіге қосылу, бірнеше процессорлар және жоғары сапалы сурет пен дыбыс енді жұмыс үстелі жүйелеріне арналған стандартқа айналды және ноутбуктер, планшеттік компьютерлер мен смартфондардың аумағын тез алады. Егер бұл үрдіс жалғаса берсе, онда барлық сұраныстарды қамтамасыз ету үшін олардың операциялық жүйелері бүгінгі жүйелерден айтарлықтай өзгеше болуы керек. Сонымен қатар, олар қуат тұтыну теңгеріміне төтеп беріп, құрылғының қолайлы температуралық режимін сақтауы керек. Жылу шығару және энергияны тұтыну тіпті жоғары деңгейлі компьютерлерде де маңызды мәселелердің бірі болып табылады. Бірақ нарықтың одан да тез дамып келе жатқан сегментін ноутбуктер, планшеттер, арзан нетбуктар мен смартфондар кіретін дербес компьютерлер құрайды. Олардың көпшілігі сыртқы әлеммен сымсыз байланысты қолдайды. Олар үшін қажет операциялық жүйесінің ерекшеленетін операциялық жүйелер үшін әзірленген құрылғылар жоғары сынып оқушылары, аз көлемі, үлкен жылдамдығымен, икемділігімен және үлкен сенімділікпен. Қазір бұл құрылғылардың көпшілігі Linux, Windows және OS X сияқты дәстүрлі операциялық жүйелерге негізделген, бірақ айтарлықтай өзгерістермен. Сонымен қатар, радиоқұралдар жиынтығын басқару үшін олар көбінесе микроядерлер мен гипервизорларға негізделген шешімдерді пайдаланады.

Бұл операциялық жүйелер қазіргі жүйелерден толық қосылуды (сым арқылы), әлсіз қосылуды (сымсыз) және желіден тыс жұмысты, соның ішінде желіден ажыратылмас бұрын деректерді жинауды және жаңа қосылудан бұрын деректердің сәйкестігін тексеруді жақсы басқаруы керек. Болашақта олар қазіргі жүйелерден гөрі ұтқырлық мәселелерін шешуге мәжбүр болады (мысалы, лазерлік принтерді табу, оған тіркелу және оған радио арқылы файл жіберу). Бұл жүйелер үшін энергияны басқару ерекше маңызды, оның ішінде операциялық жүйе мен қосымшалар арасындағы батареяларда қанша энергия қалғаны және оны қалай жақсы пайдалану керектігі туралы ұзақ диалогтар бар. Кішкентай экрандарға қосымшалардың динамикалық бейімделуі де маңызды болуы мүмкін. Сонымен, жаңа енгізу және шығару режимдері, соның ішінде, қолмен теру және сөйлеуді енгізу үшін, операциялық жүйеде сапаны жақсарту үшін жаңа әдістер қажет болуы мүмкін. Дербес қуатты

портативті сымсыз дауыспен басқарылатын компьютердің операциялық жүйесі гигабиттік талшықты-оптикалық желілік қосылымы бар 16 ядролы орталық процессоры бар 64 биттік жұмыс үстеліндегі компьютерде жұмыс істейтін жүйеден айтарлықтай өзгеше болуы мүмкін. Әрине, өз талаптары бар сансыз гибридті машиналар болады.

Кірістірілген жүйелер

Жаңа операциялық жүйелер көбейетін соңғы аймақ - кірістірілген жүйелер. Кір жуғыш машиналардағы, микротолқынды пештердегі, қуыршақтардағы, радиоқабылдағыштардағы, MP3 ойнатқыштардағы, бейнекамералардағы, лифтілердегі және кардиостимуляторлардағы операциялық жүйелер бұрын аталғандардың бәрінен және бір-бірінен өзгеше болады. Олардың әрқайсысы белгілі бір қосымшаның астына мұқият «жасырылуы» керек, өйткені оны лифт контроллеріне айналдыру үшін PCIe картасын кардиостимуляторға салу ешкімнің басына келмеуі мүмкін. Барлық ендірілген жүйелерде алдын ала белгілі бағдарламалардың шектеулі жиынтығы жұмыс істейтіндіктен, әмбебап жүйелерде оңтайландыруға тыйым салуға болады.

ПАЙДАЛАНЫЛҒАН ӘДЕБИЕТТЕР ТІЗІМІ

1. Таненбаум Э., Бос Х. Современные операционные системы. СПб.: «Питер». 4-е изд. - 2015.

2. Мэтью Нейл. Основы программирования в Linux: руководство: пер. с англ. Мэтью Нейл, Стоунс Ричард. - 4-е изд. - СПб.: БХВ-Петербург, 2009. - 596 с.: ил.

3. Мэйволд Эрик. Безопасность сетей: Практ. Пособие: Самоучитель / Мэйволд Эрик.- М.: СП ЭКОМ: БИНОМ, 2005. - 528 с.: ил. - (Шаг за шагом).

4. Олифер В.Г. Компьютерные сети. Принципы, технологии, протоколы: Учебник для Вузов. / Олифер В.Г., Олифер Н.А.- СПб. :Питер. 2010. - 943 с.: ил.

5. Басс Лен. Архитектура программного обеспечения на практике: Практ. пособие / Басс Лен, Клементс П., Кацман Р. - 2-е изд. - СПб.: Питер, 2006. - 575 с.: ил. - (Классика computer science).

6. Гордеев А.В., Молчанов А.Ю. Системное программное обеспечение. - СПб.: Питер. 2003.-736с.: ил.

7. Дейтел Харвин М. Операционные системы. Основы и принципы: Пер. с англ. - 3-е изд.- М.: БИНОМ, 2006.- 1024 с.: ил.

8. Олифер В.Г., Олифер Н.А. Сетевые операционные системы.- СПб. Питер. 2009.-672с.: ил.

9. Эви Немет, Гарт Снайдер и др. UNIX. Руководство системного администратора. - Киев: Вильямс, 2012.-1312с.

10. Курячий Г.В., Маслинский К.А. Операционная система Linux. - М-Интуит.Ру. 2005. -392 е.: ил.

11. Безбогов А.А., Мартемьянов Ю. Безопасность операционных систем, изд. Машиностроение-1, 2007г, 220с.

Оқу басылымы

БАРТОСИК ФЕЛИКС МИХАЙЛОВИЧ,
КОЖАНОВА ДИНАРА ТАЛГАТОВНА,
КАДЫРОВА ЛЯЙЛЯ БАКБЕРГЕНОВНА

ОПЕРАЦИЯЛЫҚ ЖҮЙЕЛЕР ҚАУІПСІЗДІГІ

Авторлардың редакциялығымен

Басуға қол қойылды 12.10.2021ж. Пішімі 90х60/16.
Есептік баспа табағы 4,9. Таралымы 20 дана. Тапсырыс 198.
ҚарТУ баспасы, 100027. Қарағанды, Н.Назарбаев даңғылы, 60.