Дәріс 6. Көпесептік. Өзара тұжырымдар

6.1. Операциялық жүйелердегі параллель есептеулер
6.2. Өзара тұжырымдар: бағдарламалық тәсіл
6.3. Параллельді есептеу принциптері
6.4. Өзара анықтама: аппараттық қолдау
6.5. Семафорлар
6.6. Мониторлар

6.1. Операциялық жүйелердегі параллель есептеулер

Операциялық жүйелерді дамытудағы негізгі сұрақтар процестер мен ағындарды басқарумен байланысты. Бұл ретте төмендегідей есептер шешіледі:

- Көп функциялы: бір процессорлық жүйеде бірнеше процестерді басқару.
- Мультипроцессорлық: мультипроцессорлық жүйеде бірнеше процестерді басқару.
- Таратылған есептеу: бірнеше компьютерлермен таратылған есептеу жүйесінде орындалатын процестерді басқару. Мұндай жүйелердің негізгі мысалы – кластерлер.

Бұл саланың негізгі тұжырымдамасы параллель есептеу (concurrency) тұжырымдамасы болып табылады. Параллелизм көптеген даму мәселелерін қамтиды, соның ішінде процестер арасында ақпарат алмасу, ресурстарды бөлу, процестерді синхрондау және әртүрлі процестер арасында процессордың уақытын бөлу. Бұл сұрақтар мультипроцессорлық немесе таратылған есептеу ортасында ғана емес, сонымен қатар бір процессорға негізделген көп функциялы жүйелер жағдайында да туындайды.

Параллелизм үш түрлі контексте көрінеді:

1. Бірнеше қосымшалар. Мультиаскинг бірнеше белсенді қосымшалар арасында процессордың уақытын динамикалық түрде бөлуге мүмкіндік береді.

2. Қосымшалардың құрылымы. Модульдік даму және құрылымдық бағдарламалау парадигмасының дамуы ретінде кейбір қосымшаларды параллель жұмыс істейтін көптеген процестер ретінде жасауға болады.

3. Операциялық жүйенің құрылымы. Операциялық жүйелер көбінесе процестер немесе ағындар жиынтығы ретінде жүзеге асырылады.

6.1-кестеде параллель есептеулерге қатысты кейбір негізгі терминдер келтірілген.

6.1-кесте. Параллель есептеулерге қатысты негізгі терминдер

Атомдық операция	Бөлінбейтін болып көрінетін бір немесе бірнеше нұсқаулықтардың тізбегі ретінде жүзеге асырылатын функция немесе әрекет. Нұсқаулар тізбегін бір топ ретінде орындауға кепілдік беріледі. Атомдық параллель процестерден оқшаулауды қамтамасыз етеді.
Критикалық бөлім	Ортақ ресурстарға қол жеткізуді қажет ететін және кодтың осы бөлігінде басқа процесс болған кезде орындалмауы керек процесс шеңберіндегі код бөлімі.
Өзара блоктау	Екі немесе одан да көп процестер жұмыс істей алмайтын жағдай, өйткені процестердің әрқайсысы белгілі бір әрекеттің басқа процесспен орындалуын күтеді.
Динамикалық блоктау	Екі немесе одан да көп процестер пайдалы жұмысты орындамай, басқа процестердегі өзгерістерге жауап ретінде өз күйлерін үнемі өзгертетін жағдай.
Өзара тұжырымдар	Бір процесс ортақ ресурстарға қол жеткізетін сыни учаскеде болған кезде, басқа ешқандай процесс осы ортақ ресурстардың кез-келгеніне жүгінетін сыни бөлімде бола алмайды.

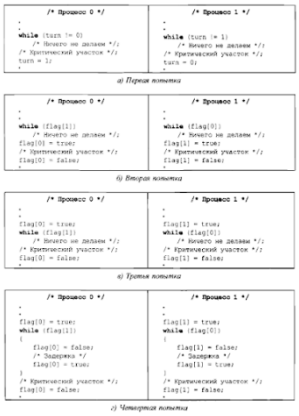
Жарыс күйі	Бірнеше ағындар немесе процестер ортақ деректер элементін оқып, жазатын жағдай және соңғы нәтиже сол ағындардың салыстырмалы орындалу уақытына байланысты болады.
Ашығу	Процесті бастауды жоспарлаушы шексіз рет өткізіп жіберетін жағдай; процесс жұмыс істеуге дайын болса да, ол ешқашан таңдалмайды

6.2. Өзара тұжырымдар: бағдарламалық тәсіл

Бағдарламалық тәсіл жалпы негізгі жады бар бір процессорда да, көп процессорлық жүйеде де орындалатын параллель процестер үшін жүзеге асырылуы мүмкін. Әдетте, мұндай тәсілдер жадқа қол жеткізу деңгейінде қарапайым өзара тұжырымдарды қамтиды. Яғни, негізгі жадтың бір ұяшығына бір уақытта қол жеткізу белгілі бір механизм арқылы реттеледі. Аппараттық құрал, операциялық жүйе немесе бағдарламалау тілі тарапынан басқа қолдау жоқ.

Деккер алгоритмі. Дейкстра [66] голландиялық математик Деккер (Dekker) ұсынған екі процестің өзара ерекшелік алгоритмі туралы хабарлады. Бұл екі орындалу ағынына тек байланыс үшін ортақ жадты қолдана отырып, кедергісіз бөлінбейтін ресурсты бөлісуге мүмкіндік береді. Егер екі процесс бір уақытта сыни бөлімге өтуге тырысса, алгоритм осы сәтте кімнің кезегіне сүйене отырып, олардың біреуіне ғана мүмкіндік береді. Егер бір процесс сыни бөлімге еніп кетсе, екіншісі біріншісінің оны тастап кетуін күтеді. Бұл екі жалаушаны (сыни бөлімге кіру ниеті индикаторларын) және turn айнымалысын (процестердің қайсысы келгенін көрсететін) пайдалану арқылы жүзеге асырылады. 6.1-суретте Деккер алгоритмін жасаудың 4 әрекеті көрсетілген.

Процестер сыни бөлімге кіру "ниеті" туралы хабарлайды; оны сыртқы "while" циклі тексереді. Егер басқа процесс мұндай ой білдірмесе, сыни бөлімге қауіпсіз кіруге болады (кімнің кезегі болса да). Өзара алып тастауға әлі де кепілдік беріледі, өйткені осы жалауды орнатпас бұрын процестердің ешқайсысы сыни бөлімге кіре алмайды (кем дегенде бір процесс "while" цикліне енеді). Бұл сонымен қатар алға жылжуға кепілдік береді, өйткені сыни бөлімге кіру үшін "ниет" қалдырған процестің күтуі болмайды. Әйтпесе, егер басқа процестің айнымалысы орнатылса, "while" цикліне енеді және turn айнымалысы кімге сыни бөлімге кіруге рұқсат етілгенін көрсетеді. Кезегі келмеген Процесс сыни бөлімге оның кезегі келгенше кіруге ниет қалдырады (ішкі цикл "while"). Кезегі келген Процесс "while" циклынан шығып, сыни бөлімге енеді.



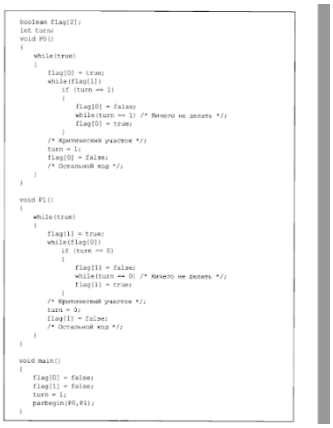
Сурет 6.1. Өзара тұжырым әрекеттері

Деккер алгоритмі 6.2 суретте көрсетілген. Parbegin дизайны (P1,P2, ...,Pn) мынаны білдіреді: негізгі бағдарламаның орындалуын тоқтата тұру; P1,P2, ...,Pn процедураларының параллель орындалуын инициализациялау; P1,P2, ...,Pn процедураларының жұмысы аяқталғаннан кейін - негізгі бағдарламаның орындалуын қалпына келтіру.

Деккер алгоритмі өзара ерекшелікке кепілдік береді. Алгоритмнің артықшылықтарының бірі-ол "тексеру - орнату" арнайы командаларын қажет етпейді - атомды оқу, модификациялау және жазу операциялары-нәтижесінде әртүрлі бағдарламалау

тілдеріне және компьютерлік архитектураларға оңай ауысады. Кемшіліктерді оның тек екі процесске қатысты қолданылуы және процесті тоқтата тұрудың орнына күту циклын қолдану деп атауға болады: күту циклын қолдану процестер сыни бөлімде ең аз уақытты өткізуі керек деп болжайды.

Қазіргі заманғы операциялық жүйелер Деккер алгоритмімен салыстырғанда қарапайым және икемді синхрондау примитивтарын ұсынады. Алайда, екі процестің арасында нақты бір мезгілде болмаған жағдайда, сыни бөлімге кіру және одан шығу операциялары осы алгоритмді қолдану кезінде өте тиімді болатындығын атап өткен жөн. Көптеген заманауи микропроцессорлар нұсқауларды рет-ретімен орындамайды, тіпті жадқа кіру тәртібі де сақталмауы мүмкін. Егер жад кедергілері қолданылмаса, алгоритм осындай процессорлармен жабдықталған SMP машиналарында жұмыс істемейді.



Сурет 6.2. Деккер алгоритмі

6.3. Параллельді есептеу принциптері

Бір процессорлы көп функциялы жүйеде процестер бір уақытта орындалудың иллюзиясын жасау үшін кезектеседі. Процестердің нақты параллель жұмысына қол жеткізілмегеніне қарамастан, сонымен қатар процестер арасында ауысуға байланысты белгілі бір үстеме шығындар бар, мұндай ауыспалы орындау бағдарламалардың тиімділігі мен құрылымы тұрғысынан айтарлықтай пайда әкеледі. Мультипроцессорлық жүйелерде процестердің ауысуы ғана емес, сонымен қатар олардың қабаттасуы да мүмкін. Параллельді есептеулердің мысалдары ретінде қарастыруға болатын ауыспалы және қабаттасу технологиялары бірдей проблемаларды тудырады. Бір процессорлық жүйелерде проблемалар көп функциялы жүйелердің негізгі сипаттамасынан туындайды: процестердің салыстырмалы жылдамдығын болжау мүмкін емес. Бұл басқа процестерге, операциялық жүйенің үзілістерін өңдеу әдісіне және операциялық жүйені жоспарлау стратегиясына байланысты. Бұл жағдайда келесі қиындықтар туындайды:

1. Жаһандық ресурстарды бөлісу қауіпті.
2. Операциялық жүйеге ресурстарды бөлуді оңтайлы басқару қиын.
3. Бағдарламалық жасақтама қатесін анықтау өте қиын болады, өйткені әдетте бағдарламаның нәтижесі анықталмайды және ойнатылмайды.

Мультипроцессорлық жүйеде бұл проблемалар бір процессорлық жүйемен бірдей. Келесі процедураны қарастырайық:

```
void echo ()
chin = getchar();
chout = chin;
putchar(chout);
```

Бұл енгізілген таңбалардың негізгі дисплей элементтерін көрсетеді. Пернетақтадан алынған кіріс таңбасы chin айнымалысында сақталады; осыдан кейін chin айнымалы мәні chout айнымалысына тағайындалады және экранға шығарылады. Бұл процедураны

пайдаланушының кірісін алу және оны экранда көрсету үшін кез-келген процесс бірнеше рет шақыра алады.

Бізде бір пайдаланушыны қолдайтын бірпроцессорлы көп тапсырмалы жүйе бар деп елестетіп көрейік. Қарастырылып отырған процедура барлық қолданбаларға қажет болғандықтан, оны барлық қолданбалар үшін ғаламдық жад бөлігіне жүктелетін ортақ процедураға айналдыру мағынасы бар. Процестер арасында негізгі жадты ортақ пайдалану процестер арасындағы тиімді және тығыз байланысты қамтамасыз етеді. Дегенмен, бұл бөлісу қиындықтарға әкелуі мүмкін.

Процестер 6.3-суретте көрсетілген реттілікпен жүзеге асырылған кезде, P1 процесімен енгізілген таңба дисплейден бұрын жоғалады, ал P2 процесі оқитын символ екі процесспен де шығады.

6.3-сурет. Ортақ глобальді айнымалылары бар процестерді іске қосу

Бір процессорлық жүйеде мәселенің себебі-үзіліс процестің ерікті жерде орындалуын тоқтата алады. Мультипроцессорлық жүйеде жұмыс жағдайлары бірдей, бірақ мәселе бір уақытта орындалатын екі процесс бір уақытта бірдей глобальді айнымалыға ауыса алатындығына байланысты туындауы мүмкін. Алайда, екі типтегі мәселелерді шешу бірдей: ортақ ресурстарға қол жеткізуді басқару.

Жарыс күйі. Жарыс күйі бірнеше процестер немесе ағындар деректер элементтерін оқып, жазған кезде пайда болады, осылайша түпкілікті нәтиже бірнеше процестердегі нұсқауларды орындау тәртібіне байланысты болады. Мысал ретінде, екі процесс, P_1 және P_2, ғаламдық айнымалы *a*-ны бөліседі делік, белгілі бір уақытта P_1 *a* мәнін жаңартады, оны 1-ге тең етеді, ал P_2 процесі оны орындау кезінде *a* мәнін жаңартады, оны 2-ге тең етеді. Осылайша, екі тапсырма "жарыста" тұр, оның жүлдесі айнымалы жазба болып табылады. Бұл мысалда "жеңілген" жарыс процесі (айнымалы соңғысын жаңартады) *a* айнымалысының соңғы мәнін анықтайды.

Операциялық жүйенің қатысуы. Параллельді есептеулердің болуына байланысты туындайтын операциялық жүйелерді жобалау мен басқарудың келесі сұрақтарын тізімдеуге болады.

1. Амалдық жүйе әртүрлі процестерді бақылай алуы керек.
2. Операциялық жүйе әр белсенді процесс үшін әртүрлі ресурстарды, соның ішінде мыналарды бөліп, босатуы керек.

- Процессорлық уақыт.

- Жад.
- Файлдар.
- Енгізу-шығару құрылғылары.

3. Операциялық жүйе әр процестің деректері мен физикалық ресурстарын жадпен, файлдармен және енгізу-шығару құрылғыларымен жұмыс істеу үшін қолданылатын технологияларды қолдануды қамтитын басқа процестердің кездейсоқ әсерінен қорғауы керек.

4. Процестің жұмыс істеуі және оның жұмысының нәтижесі параллель орындалатын басқа процестерге қатысты оның орындалу жылдамдығына байланысты болмауы керек. Бұл тарау осы мәселеге арналған.

Процестердің салыстырмалы орындалу жылдамдығынан тәуелсіздігі мәселелерін жақсы түсіну үшін алдымен процестердің өзара әрекеттесу тәсілдерін қарастырайық.

Процестердің өзара әрекеттесуі. Процестердің өзара әрекеттесу тәсілдерін бір процестің екіншісінің бар екендігі туралы хабардар болу дәрежесі бойынша жіктеуге болады. 6.2-кестеде хабардарлықтың үш мүмкін дәрежесі көрсетілген.

- Процестер бір-бірінің болуы туралы білмейді. Бұл тәуелсіз процестер, олар бірлесіп жұмыс істеуге арналмаған. Мұндай жағдайдың ең жақсы мысалы көптеген тәуелсіз процестердің көп болуы болуы мүмкін. Бұл пакеттік тапсырмалар, интерактивті сессиялар немесе екеуінің тіркесімі болуы мүмкін. Бұл процестер бірлесіп жұмыс жасамаса да, Операциялық жүйе ресурстарды бәсекеге қабілетті пайдалану мәселелерін шешуі керек.
- Процестер бір-бірінің болуы туралы жанама түрде біледі. Бұл процестер процесс идентификаторына дәл сәйкес келетін бір-бірінің бар екендігі туралы міндетті түрде хабардар болмауы керек, бірақ олар кейбір объектіге, мысалы, енгізу-шығару буферіне жүгінеді. Мұндай процестер ортақ объектіні бірлесіп пайдалану кезіндегі ынтымақтастықты көрсетеді.
- Процестер бір-бірінің болуы туралы тікелей біледі. Мұндай процестер процесс идентификаторларын қолдана отырып, бір-бірімен байланыса алады және бастапқыда бірлесіп жұмыс істеу үшін жасалады. Олар сондай-ақ жұмыс кезінде ынтымақтастықты көрсетеді.

6.2 -кесте. Процестердің өзара әрекеттесуі

Хабардарлық дәрежесі	Өзара байланыс	Бір процестің екіншісіне әсері	Ықтимал проблемалар
Процестер бір-бірін білмейді	Бәсекелестік	• Бір процестің нәтижесі басқалардың әрекеттеріне байланысты емес • Бір процестің екіншісінің жұмыс уақытына әсер етуі мүмкін	• Өзара тұжырым • Блоктау (жаңартылатын ресурстар) • Ашығу
Процестер жанама түрде бір-бірін біледі	Ортақ ресурстарды пайдаланумен ынтымақтастық	• Бір процестің нәтижесі басқа процестерден алынған ақпаратқа байланысты болуы мүмкін • Бір процестің екіншісінің жұмыс уақытына әсер етуі мүмкін	• Өзара тұжырым • Блоктау (жаңартылатын ресурстар) • Ашығу • Деректер байланысы

Процестер бір-бірін тікелей біледі	Байланысты пайдаланумен ынтымақтастық	• Бір процестің нәтижесі басқалардан алынған ақпаратқа байланысты болуы мүмкін • Бір процестің екіншісінің жұмыс уақытына әсері болуы мүмкін ресурстар үшін күрестегі бәсекелестік процестер	• Өзара блоктау (жұмсалатын ресурстар) • Ашығу

Процестердің өзара әрекеттесуінде операциялық жүйе үш негізгі проблемаға тап болады. Олардың біріншісі-өзара ерекшеліктердің қажеттілігі (mutual exclusion). Екі немесе одан да көп процестер принтер сияқты бір бөлінбейтін ресурсқа қол жеткізуді қажет етеді делік. Орындау кезінде әр процесс енгізу-шығару құрылғысына пәрмендер жібереді, оның күйі туралы ақпарат алады, деректерді жібереді және/немесе алады. Біз сыни ресурс сияқты ресурс туралы, ал оны қолданатын бағдарламаның бөлігі - бағдарламаның сыни бөлімі (critical section) туралы айтатын боламыз. Кез-келген уақытта сыни аймақта тек бір бағдарлама болуы өте маңызды. Біз Операциялық жүйе жағдайды таниды және осы Шартты орындайды деп сене алмаймыз, өйткені ресурсқа қойылатын толық талаптар айқын болмауы мүмкін.

Өзара ерекшеліктерді жүзеге асыру екі қосымша проблема туғызады. Олардың бірі-өзара құлыптау (deadlock). Мысалы, екі процесті (P1 және P2) және екі ресурсты (R1 және R2) қарастырыңыз. Әрбір процесс өз функцияларының бір бөлігін орындау үшін екі ресурстарға қол жеткізуді қажет етеді делік. Содан кейін келесі жағдай туындауы мүмкін: Операциялық жүйе R1 ресурсын P2 процесіне, ал R2 ресурсын P1 процесіне бөледі. Нәтижесінде әр процесс екі ресурстың біреуін алады деп күтеді; сонымен қатар, олардың ешқайсысы екі ресурстың болуын талап ететін функцияларды орындау үшін екінші ресурсты алуды күтіп, өзінде бар ресурсты босатпайды. Нәтижесінде процестер өзара бұғатталады.

Соңғы мәселе - ашығу (starvation). Бізде үш процесс бар делік (P1, P2, P3), олардың әрқайсысы R ресурсына мезгіл-мезгіл қол жеткізуді қажет етеді. P1 сыни бөлімнен шыққаннан кейін ресурсқа P2 және P3 процестерінің бірі қол жеткізеді. Операциялық жүйе R ресурсына P3 процесіне қол жеткізуге мүмкіндік берсін. Ол ресурспен жұмыс істеп тұрған кезде, ресурсқа кіру қайтадан P1 процесін қажет етеді. Нәтижесінде, P3 процесі ресурстарды босатқаннан кейін, Операциялық жүйе P1 процесіне ресурсқа қайта қол жеткізуді қамтамасыз етуі мүмкін; осы уақытта P3 процесі R ресурсына қайта кіруді қажет етеді, сондықтан P2 процесі ешқашан қажет ресурсқа қол жеткізе алмайтын жағдай теориялық тұрғыдан мүмкін, дегенмен бұл жерде өзара құлыптау жоқ жоқ жағдайда.

Өзара алып тастауға қойылатын талаптар. Өзара ерекшеліктерді қолдауды қамтамасыз етудің кез келген мүмкіндігі мынадай талаптарға сәйкес келуі тиіс.

1. Өзара бас тарту мәжбүрлеу тәртібімен жүзеге асырылуға тиіс. Кез-келген уақытта, бір ресурс немесе ортақ объект үшін сыни учаскесі бар барлық процестердің ішінде тек бір ғана процесс болуы мүмкін.
2. Сыни емес аймақта жұмысты аяқтайтын процесс басқа процестерге әсер етпеуі керек.
3. Сыни аймаққа кіруді шексіз күту жағдайы болмауы керек (яғни, блоктау және аштық болмауы керек).
4. Егер сыни аймақта бірде-бір процесс болмаса, оған кіру мүмкіндігін сұраған кез-келген процесс оны дереу алуы керек.
5. Процестердің саны немесе олардың салыстырмалы жұмыс жылдамдығы туралы ешқандай болжам жасалмайды.

6. Процесс сыни аймақта шектеулі уақытқа ғана қалады.

Аталған шарттарды қанағаттандырудың бірқатар жолдары бар. Олардың бірі-талаптарға сәйкестік үшін жауапкершілікті параллель орындалуы керек процестің өзіне беру. Бұл мәселе алдыңғы бөлімде қаралды. 6.4-бөлімде қарастырылған тағы бір тәсіл арнайы мақсаттағы машиналық командаларды қолдануды қамтиды. Бұл тәсілдің артықшылығы-шығындарды азайту, бірақ бұл тәсіл жалпы жағдайда мәселені шешпейді. Тағы бір тәсіл-операциялық жүйенің немесе бағдарламалау тілінің белгілі бір қолдау деңгейін қамтамасыз ету. Өзара тұжырымдар мәселесін шешудің осы тәсілінің маңызды нұсқалары 6.5-5.7 бөлімдерінде қарастырылады.

6.4. Өзара анықтама: аппараттық қолдау

Бұл бөлімде біз өзара қорытынды мәселесін шешудің бірнеше қызықты аппараттық тәсілдерін қарастырамыз.

Үзілістерді өшіру. Машинада бір ғана процессор болған кезде, параллель процестер бір-біріне сәйкес келе алмайды, олар тек ауыса алады. Сонымен қатар, процесс Операциялық жүйе қызметі шақырылғанға дейін немесе процесс тоқтатылғанға дейін жалғасады. Демек, өзара алып тастауға кепілдік беру үшін процесті үзілістен қорғау жеткілікті. Бұл мүмкіндікті үзілістерге тыйым салу және шешу үшін операциялық жүйенің өзегі анықтаған примитивтер түрінде қамтамасыз етуге болады. Бұл жағдайда процесс келесі түрде өзара анықтама бере алады:

while (true} /* үзілулерді тыю*/;

/ * Сыни аймақ*/;

/ * үзілулерге рұқсат алу */;

/ * Қалған код */;

Алайда, бұл тәсілдің бағасы жоғары. Жұмыс тиімділігі айтарлықтай төмендеуі мүмкін, өйткені процессордың бағдарламаларды ауыстыру мүмкіндігі шектеулі. Тағы бір мәселе, бұл тәсіл мультипроцессорлық архитектурада жұмыс істемейді.

Арнайы машина командалары. Аппараттық деңгейде, жоғарыда айтылғандай, жад ұяшығына жүгіну сол ұяшыққа кез-келген басқа қоңырауды болдырмайды. Осы принципке сүйене отырып, процессорды жасаушылар бірнеше машиналық командаларды ұсынады, олар бір іріктеу циклінде командалар жад ұяшығында екі әрекетті атомдық түрде орындайды, мысалы, оқу және жазу немесе оқу және мәнді тексеру. Бұл әрекеттер бір іріктеу циклінде орындалатындықтан, оларға басқа нұсқаулар әсер ете алмайды.

Салыстыру және тағайындау командасы. Салыстыру және тағайындау командасы, compare&swap, келесідей анықталуы мүмкін [104]:

```
int compare_and_swap(int* word, int testval, int newval)
{
int oldval;
oldval = *word;
if (oldval == testval) *word = newval;
return oldval;
}
```

Команданың бұл нұсқасы жад ұяшығын (*word) тексереді, оның мәнін тестпен (testval) салыстырады. Егер жад ұяшығының ағымдағы мәні testval болса, ол newval мәнімен ауыстырылады; әйтпесе жад ұяшығының мәні өзгеріссіз қалады. Ескі жад ұяшығының мәні әрқашан қайтарылады; осылайша, егер қайтару мәні тестке сәйкес келсе, жад ұяшығы жаңартылады. Бұл атом командасы екі бөліктен тұрады (жад ұяшығындағы мәндер мен сынақ мәнін салыстыру) және егер бұл мәндер сәйкес келсе, тағайындау орындалады. Барлық функция атомдық түрде орындалады, яғни. ол мүмкін емес жоғалтады. 6.3-суретте сипатталған команданы қолдануға негізделген өзара алып тастау ХАТТАМАСЫ көрсетілген.

Бірлескен айнымалы bolt 0 мәнімен бапталады. Bolt мәні 0-ге тең екенін анықтайтын процесс ғана критикалық бөлімге ене алады. Сыни аймаққа кіруге тырысатын барлық басқа процестер жұмыспен қамтуды күту режиміне өтеді.

```
/* program mutualexclusion */
const int n = /* Количество процессов */;
int bolt;
void P(int i)
{
  while (true)
  {
    while(compare_and_swap(&bolt, 0, 1)
          == 1)
          /* Ничего не делать */;

    /* Критический участок */;
       bolt = 0;
    /* Остальной код */;
  }
}
void main()
{
   bolt = 0;
   parbegin(P(1), P(2), ... , P(n));
}
```

```
/* program mutualexclusion */
int const n = /* Количество процессов */;
int bolt;
void P(int i)
{
  while (true)
    int keyi = 1;
    do exchange(&keyi, &bolt)
    while (keyi != 0);
    /* Критический участок */;
    bolt = 0;
    /* Остальной код */;
  }
}

void main()
{
   bolt = 0;
   parbegin(P(1), P(2), ..., P(n));
}
```

Сурет 6.3. Өзара қорытындыларды аппараттық қолдау

Жұмыспен қамтуды күту (busy waiting, spin waiting) термині оған сәйкес процесс сыни аймаққа кіруге рұқсат алғанға дейін кіруге рұқсат алу үшін тиісті айнымалыны тексеру жөніндегі нұсқаулықты орындаудан басқа ештеңе істей алмайтын Әдістемеге жатады. Процесс критикалық бөлімнен шыққан кезде, ол bolt айнымалы мәнін 0-ге қалпына келтіреді; осы кезде күтілетін процестердің біреуі және біреуі ғана критикалық аймаққа қол жеткізеді. Процесті таңдау салыстыру және тағайындау пәрменін қай процестің бірінші орындайтынына байланысты.

Бөлісу командасы. Бөлісу тобын келесідей анықтауға болады:

```
void exchange(int* register, int* memory)
{
int temp;
temp = *memory;
*memory = *register;
*register = temp;
}
```

Бұл пәрмен регистр мен жад ұяшықтарының мазмұнын бөліседі. XCHG командасы Intel ia-32 (Pentium) архитектурасында да, IA-64 (itanium) архитектурасында да бар.
5.5-б суретте осы команданы қолдануға негізделген өзара fлып тастау ХАТТАМАСЫ көрсетілген.

Өзара ерекшеліктерді жүзеге асыру үшін арнайы машиналық нұсқаулықты қолдануға негізделген тәсіл бірқатар артықшылықтарға ие.

- Негізгі жадты бөлісетін бір және бірнеше процессорлар болған кезде процестердің кез-келген санына қолданылады.
- Өте қарапайым, сондықтан оны тексеру оңай.
- Көптеген сыни аймақтарды қолдау үшін пайдаланылуы мүмкін; олардың әрқайсысын өз айнымалысы арқылы анықтауға болады.

  Алайда, бұл тәсілдің елеулі кемшіліктері бар.
- Жұмыспен қамтуды күту қолданылады. Демек, процесс сыни аймаққа кіруді күтіп тұрған кезде, ол процессордың уақытын тұтынуды жалғастыруда.
- Ашығу болуы мүмкін. Егер процесс сыни аймақты тастап кетсе және оған бірнеше басқа процестер кірсе, онда күтілетін процесті таңдау ерікті болып табылады. Демек, кейбір процестер сыни аймаққа шексіз кіруді күтуі мүмкін.

- Блоктау болуы мүмкін. Бағдарламалық және аппараттық шешімдерді қолдану кезінде кемшіліктердің болуына байланысты біз бұғаттауды қамтамасыз етудің басқа тетіктерін қарастыруымыз керек.

6.5. Семафорлар

Параллельді есептеуді қамтамасыз ететін іргелі принцип-бұл екі немесе одан да көп процестер қарапайым сигналдар арқылы жұмыс істей алады, сондықтан белгілі бір жерде процесс тиісті сигнал күтілгенге дейін жұмысын тоқтата алады. Күрделіліктің кез келген деңгейіндегі кооперацияның талаптары сигналдардың тиісті құрылымымен қанағаттандырылуы мүмкін. Сигнал беру үшін семафорлар деп аталатын арнайы айнымалылар қолданылады. S семафоры арқылы сигнал беру үшін процесс semsignal(s) примитивін орындайды, ал оны алу үшін semwait(s) примитивін орындайды. Соңғы жағдайда процесс тиісті сигнал берілгенге дейін тоқтатылады.

Семафорды үш операция анықталған бүтін мәні бар айнымалы ретінде қарастыруға болады.

1. Семафорды теріс емес бүтін мәнге инициализациялауға болады.

2. Semwait операциясы семафордың мәнін төмендетеді. Егер бұл мән теріс болса, semWait операциясын орындау процесі бұғатталады.

3. SemSignal операциясы семафордың мәнін арттырады. Егер бұл мән нөлден аз немесе оған тең болса, semWait операциясымен бұғатталған процесс (бар болса) бұғатталған.

Көрсетілгендерден басқа семафордың мәні туралы ақпарат алудың немесе оның мәнін өзгертудің басқа тәсілдері жоқ.

Жұмыстың басында семафордың нөлдік мәні немесе кейбір оң мәні бар. Егер мән оң болса, онда ол сигнал алу әрекетін тудыруы мүмкін және дереу орындауды жалғастыратын процестердің санына тең. Егер мән нөлге тең болса (инициализация кезінде алынған немесе семафордың бастапқы мәніне тең процестер саны күту операциясынан туындаған), келесі күту процесі бұғатталады және семафордың мәні теріс болады. Әрбір келесі күту семафордың мәнін төмендетеді, осылайша ол теріс мәнге ие болады, модуль құлыпты ашуды күтетін процестердің санына тең. Семафордың мәні теріс болған кезде, әр сигнал күтілетін процестердің бірін ашады.

6.4-суретте семафор примитивтерінің неғұрлым ресми анықтамасы келтірілген. Semwait және setSignal примитивтері атомдық, яғни оларды үзуге болмайды деп болжанады. Екілік семафор ретінде белгілі семафордың неғұрлым шектеулі нұсқасы 6.5-суретте көрсетілген. Екілік семафор тек 0 немесе 1 мәндерін қабылдай алады және оны келесі үш операция арқылы анықтауға болады.

1. Екілік семафорды 0 немесе 1 мәнімен баптауға болады.

2. SemWaitB операциясы семафордың мәнін тексереді. Егер бұл мән нөлге тең болса, semWaitB орындайтын процесс бұғатталады. Егер мән 1 болса, ол өзгереді, 0-ге тең болады және процестің орындалуы жалғасады.

3. SemSignalB операциясы осы семаформен бұғатталған процестің бар-жоғын тексереді (семафордың мәні 0-ге тең). Егер бар болса, semWaitB операциясымен бұғатталған процесс құлыптан босатылады. Егер бұғатталған процестер болмаса, семафордың мәні 1-ге тең болады.

Сурет 6.4. Семафор примитивтерін анықтау

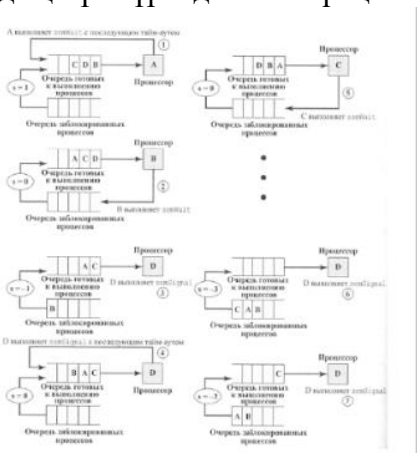
Сурет 6.5. Екілік семафордың примитивтерін анықтау

Мутекс ұғымы екілік семаформен тығыз байланысты (өзара ерекше құлыптау - mutual exclusion lock (mutex)). Мутекс-бұл нысанды басып алу және босату үшін қолданылатын бағдарламалық жасақтама жалауы. Жалпы болуы мүмкін емес деректерді түсірген кезде немесе жүйенің басқа жерлерінде бір уақытта орындалмайтын өңдеуді бастаған кезде, мутекс оны қолданудың басқа әрекеттерін болдырмайтын құлыптау күйіне орнатылады (әдетте 0). Мутекс деректер қажет болмаған кезде немесе процедура аяқталған кезде құлыптан босатылады. Мутекс пен екілік семафордың басты айырмашылығы-мутекске тосқауыл қою (оның мәнін нөлге қою) процесі оның құлпын ашумен бірдей болуы керек (оның мәнін 1-ге қою). Екілік Семафор жағдайында оны бір процесс бұғаттай алады, ал екіншісін бұғаттай алады.

Кәдімгі және екілік семафорларды күтетін процестерді сақтау үшін кезек қолданылады. Бұл жағдайда процестерді осы кезектен алу тәртібі туралы мәселе туындайды. Ең дұрыс әдіс - "бірінші кіру-бірінші шығу" принципін қолдану (first-in-first-out - FIFO). Сонымен қатар, бірінші кезекте басқаларға қарағанда ұзағырақ бұғатталған процесс босатылады. Бұл әдісті қолданатын Семафор күшті Семафор (*strong semaphore*) деп аталады. Кезектен процестерді алу тәртібі анықталмаған Семафор әлсіз Семафор деп аталады (*weak semaphore*). 6.6-суретте күшті семафордың жұмысының мысалы көрсетілген. Мұнда A, B және C процестері d процесінің нәтижелеріне байланысты болады, бастапқыда a процесі жұмыс істейді (1 кезең); B, C және D процестері өз кезегін күтіп, белсенді процестер тізімінде болады. Семафордың мәні l, бұл D процесінің нәтижелерінің бірі қол жетімді екенін көрсетеді. A процесі *semWait* нұсқауларын орындағанда, семафор 0-ге дейін төмендейді, ал A процесі жалғасуда және кейінірек белсенді процестер тізімінде орындалу кезегіне айналады. Содан кейін в процесі (2 кезең) басталады, ол *semWait* нұсқауларын да орындайды, нәтижесінде процесс тоқтап, d

процесін бастауға мүмкіндік береді (3 кезең). D процесі жаңа нәтиже алу үшін жұмысты аяқтаған кезде, ол *semSignal* нұсқаулығын орындайды, ол B процесін тоқтатылған процестер тізімінен белсенділер тізіміне өтуге мүмкіндік береді (4 кезең). D процесі белсенді процестердің кезегіне қосылады және C процесі (5 кезең) іске қосылады, бірақ *semWait* нұсқауларын орындау кезінде бірден тоқтатылады. Сол сияқты, A және B процестерінің орындалуы да тоқтатылады, бұл D процесін бастауға мүмкіндік береді (6 кезең). D процесінің жаңа нәтижесі алынғаннан кейін, олар semSignal нұсқаулығын орындайды, ол C процесін тоқтатылған тізімнен белсенді процестер тізіміне ауыстырады. D процесінің келесі циклдары A және B процестерін ажыратады.

Келесі бөлімде өзара алып тастау алгоритмі (5.9-сурет) қарастырылады, мұнда күшті семафорды пайдалану аштыққа кепілдік бермейді, ал әлсіз семафора бұлай болмайды. Келесіде біз күшті семафорлар жұмыс істейді деп есептейміз, өйткені олар ыңғайлырақ және семафордың бұл түрі әдетте операциялық жүйеде қолданылады.



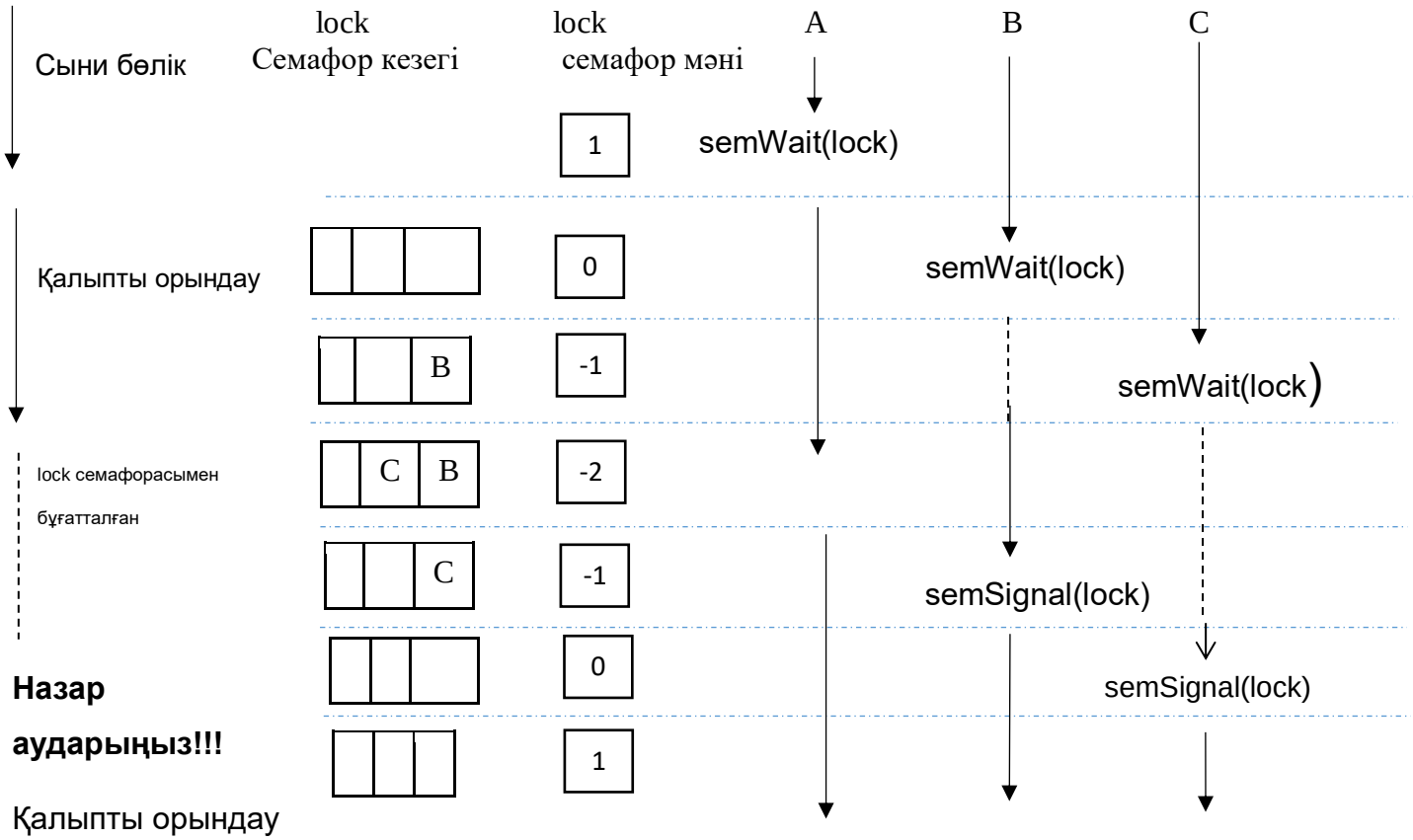
Сурет 6.6. Семафордың жұмысының мысалы

```
/*program mutualexclusion*/
const int n=/*Процессорлар саны*/;
 semaphore s=1;
void P{int i}
{
while{true}
{
semWait(s);
/*Сыни бөлік*/;
semSignal(s);
/*Қалған код*/;
}
}
Void main()
{
Parbegin(P(1),P(2),…, P(n));
}
```

Сурет 6.7. Семафорларды пайдалана отырып өзара бас тарту

6.7-суретте s семафорын қолдана отырып, өзара тұжырымдар мәселесінің қарапайым шешімі көрсетілген (5.4 суретпен салыстырыңыз). Бізде P(i) массивімен анықталатын N процестер болсын. Сыни бөлімге кірмес бұрын процестердің әрқайсысында semwait(s) қоңырауы орындалады. Егер s мәні теріс болса, процесс тоқтатылады. Егер мән 1-ге тең болса, ол нөлге дейін азаяды және процесс бірден критикалық бөлімге енеді; s енді оң емес болғандықтан, басқа ешқандай процесс критикалық бөлімге кіре алмайды.

6.8-суретте ([15)) 6.7-суретте келтірілген өзара алып тастау технологиясын пайдалану кезінде үш процестің ықтимал әрекеттер тізбегі көрсетілген. Бұл мысалда үш процесс (A, B, C) lock семафорымен қорғалған ортақ ресурсқа жүгінеді.



Назар аударыңыз!!!

Қалыпты орындау параллель болуы мүмкін, бірақ сыни бөлім – тек тізбектей.

Сурет 6.8. Процестердің семаформен қорғалған жалпы мәліметтерге қол жетімділігі

6.6. Мониторлар

Семафорлар өзара ерекшеліктер мен процестерді үйлестіру үшін жеткілікті күшті және икемді құрал ұсынады.

Дегенмен, 5.12-суретте көргеніңіздей, семафорлардың көмегімен дұрыс жұмыс істейтін бағдарламаны құру әрқашан оңай емес. Қиындық-semWait және semSignal операциялары бүкіл бағдарламаға таралуы мүмкін және олардың бақыланатын семафорларға әсерін бірден бақылау әрдайым мүмкін емес.

Монитор-бұл семафорлардың функционалдығына тең, бірақ басқаруға оңай болатын функционалдылықты қамтамасыз ететін бағдарламалау тілінің дизайны. Мониторлар тұжырымдамасының ресми анықтамасы алғаш рет [107] берілген.

Мониторлар көптеген бағдарламалау тілдерінде, соның ішінде Concurrent Pascal, Pascal-Plus, Modula-2, Modula-3 және Java тілдерінде орындалады. Мониторлар бағдарламалық кітапханалар ретінде де жүзеге асырылады. Бұл кез-келген нысанды блоктайтын мониторларды пайдалануға мүмкіндік береді. Атап айтқанда, мысалы, байланыстырылған тізім үшін сіз барлық байланыстырылған тізімдерді бір құлыппен бұғаттай аласыз немесе әр тізім үшін, мүмкін тізімнің әр элементі үшін бөлек құлыптарға ие бола аласыз.

Монитор-бұл инициализация тізбегінен, бір немесе бірнеше процедуралардан және жергілікті деректерден тұратын бағдарламалық модуль.

Монитордың негізгі сипаттамалары келесідей.

1. Монитордың жергілікті айнымалылары тек оның процедураларына қол жетімді; монитордың жергілікті деректеріне қол жеткізудің сыртқы процедуралары жоқ.
2. Процесс мониторға оның процедураларының бірін шақыру арқылы кіреді.
3. Мониторда белгілі бір уақытта тек бір процесс орындалуы мүмкін; мониторды тудырған кез-келген басқа процесс монитордың қол жетімділігін күту кезінде тоқтатылады.

Белгілі бір уақытта тек бір процесті орындау шарттарын сақтау мониторға өзара анықтама беруге мүмкіндік береді. Монитор деректері осы кезде тек бір процесте қол жетімді, сондықтан ортақ деректер құрылымын мониторға қою арқылы қорғауға болады. Егер монитордағы деректер қандай да бір ресурсты білдірсе, онда монитор ресурсқа жүгінген кезде өзара алып тастауды қамтамасыз етеді.

Параллельді есептеулерде кеңінен қолдану үшін мониторларда синхрондау құралдары болуы керек. Мысалы, процесс мониторды шақырады және мониторда болу белгілі бір шарт орындалғанға дейін тоқтатылуы керек делік. Бұл жағдайда бізге процесті тоқтатып қана қоймай, мониторды босатып, оған басқа процеске кіруге мүмкіндік беретін белгілі бір механизм қажет. Кейінірек, шарт орындалып, монитор қол жетімді болған кезде, тоқтатылған процесс өз жұмысын тоқтатылған жерден жалғастыра алады.

Мониторда бар және тек қол жетімді шартты айнымалыларды (condition variable) пайдаланып синхрондауды қолдайды. Екі функция осы айнымалылармен жұмыс істей алады.

- *cwait(c).* С шарты бойынша шақырушы процестің орындалуын тоқтатады, бұл ретте Монитор басқа процесте пайдалану үшін қол жетімді болады.
- *csignal(c).* Cwait шақыруымен сол шартпен тоқтатылған кейбір процестің орындалуын қалпына келтіреді. Егер бірнеше осындай процестер болса, олардың біреуі таңдалады; егер мұндай процестер болмаса, функция ештеңе істемейді.

Монитордың wait/signal операциялары тиісті Семафор операцияларынан өзгеше екенін ескеріңіз. Егер монитордағы процесс сигнал берсе, бірақ оны күткен бірде-бір процесс болмаса, онда сигнал жоғалады.

6.9-суретте монитордың құрылымы көрсетілген. Процесс мониторға оның кез келген процедурасын шақыру арқылы кіре алатынына қарамастан, біз мониторды кез келген уақытта мониторда ең көбі бір процесс болуын қамтамасыз ететін жалғыз кіру нүктесі ретінде қарастырамыз. Мониторға кіруге әрекеттенетін басқа процестер монитор қолжетімді болғанша тоқтатылған процестердің кезегіне қосылады. Процесс мониторға кіргеннен кейін, *cwait(x)*қоңырауын орындау арқылы x шартын күту үшін уақытша тоқтатылуы мүмкін; осыдан кейін процесс шарт орындалған кезде мониторға қайта кіруді күтетін процестердің кезегіне орналастырылады және *cwait(x)* қоңырауынан кейінгі бағдарламаның нүктесінде жұмысын жалғастырады.

Егер мониторда орындалатын процесс өзгермелі жағдайлардың өзгеруін анықтаса, ол тиісті кезектің анықталған өзгерісі туралы есеп беретін *csignal(x)* операциясын орындайды.

Сур. 6.9. Монитор құрылымы

ҚОЛДАНЫЛҒАН ӘДЕБИЕТТЕР ТІЗІМІ

1. Garg, R.; Verma, G.Operating Systems [OP]: An Introduction - Softcover Publisher: Mercury Learning & Information, 2017. 290 p.
2. https://gifer.com/ru/7h0m
3. https://3dnews.ru/1034959
4. Darrell Hajek, Cesar Herrera, Flor Narciso Principles of Operating Systems. Independently Published (24 April 2020) 176 pages.
5. Andrew S. Tanenbaum and Herbert Bos. Modern Operating Systems. 4/E.1136 pages, Pearson India, 2016.
6. Silberschatz Abraham, Galvin Peter Baer and Gadne Greg. Operating system concepts.
7. Amdahl GM (1967) Validity of the single-processor approach to achieve large scale computing capabilities. AFIPS Joint Spring Conference Proceedings 30 (Atlantic City, NJ, Apr. 18–20), AFIPS Press, Reston VA, pp 483–485.
8. https://studfile.net/.
9. https://habr.com/ru/post/40227/.
10. wikimedia.org
11. wordpress.com
12. blackandwhitecomputer.blog
13. http://www-inst.eecs.berkeley.edu/~n252/paper/Amdahl.pdf.
14. encyclopedia2.thefreedictionary.com
15. linustechtips.com
16. youtube.com/watch?v=w3K1JklY6D4