13.4 МУЛЬТИПРОЦЕСОР ІЗ ЗАГАЛЬНОЮ ПАМ'ЯТТЮ

Мультипроцесор із загальною пам'яттю (shared-memory multiprocessor, далі просто мультипроцесор) – комп'ютерна система, в якій два і більше центральних процесора мають повний доступ до загальної оперативної пам'яті. Програма, запущена на будь-якому з центральних процесорів, бачить звичайний віртуальний простір, що має, як правило, сторінкову організацію. Єдина незвичайна властивість, притаманна цій системі, полягає в тому, що центральний процесор може записати якесь значення в слово пам'яті, а потім зчитати це слово і отримати інше значення, тому що інший центральний процесор його вже змінив. При належній організації ця властивість формує основу для міжпроцесорного обміну даними: один центральний процесор записує якісь дані в пам'ять, а інший їх зчитує з пам'яті.

Здебільшого мультипроцесорні операційні системи мало чим відрізняються від звичайних. Вони обробляють системні виклики, здійснюють управління пам'яттю, надають файлову систему і керують пристроями введення- виведення. Проте є ряд областей, де вони мають унікальні особливості. Ці області включають синхронізацію процесів, управління ресурсами і планування. Далі ми спочатку дамо короткий огляд мультипроцесорного апаратного забезпечення, а потім перейдемо до питань, що стосуються операційних систем.

13.4.1 Мультипроцесорне апаратне забезпечення

Для простої і «дешевої» підтримки багатопроцесорної організації була запропонована архітектура SMP (Symmetric Multi-Processors) – мультипроцесування з розподілом пам'яті, що припускає об'єднання процесорів на загальній шині оперативної пам'яті (рис. 13.4). Слово «симетричний» в назві архітектури означає, що кожен процесор може робити все те, що і будь-який інший процесор. За апаратну простоту реалізації засобів SMP доводиться розплачуватися процесорним часом очікування в черзі до шини оперативної пам'яті. Коли CPU хоче прочитати слово в пам'яті, він спочатку перевіряє, чи вільна шина. Якщо шина вільна, CPU виставляє на неї адресу потрібного йому слова, подає кілька керуючих сигналів і чекає, поки пам'ять не виставить потрібне слово на шину даних.

Якщо шина зайнята, CPU просто чекає, поки вона не звільниться. У цьому полягає проблема даної архітектури. При двох або чотирьох CPU змаганням за шину

можна ще управляти. Але при 32 або 64 CPU шина буде постійно зайнята, а продуктивність системи буде повністю обмежена пропускною здатністю шини. При цьому більшу частину часу CPU будуть простоювати.

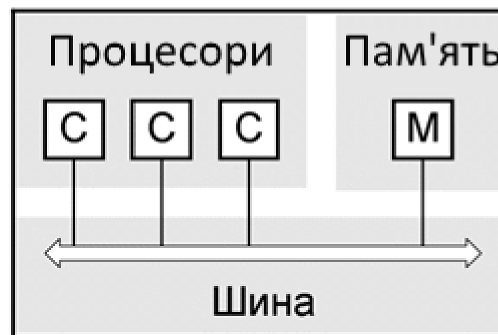

Рисунок 13.4 – Мультипроцесорна система із загальною шиною

У більшості випадків користувачі готові додати в сервер один або більше процесорів (але рідко – більше чотирьох) в надії збільшити продуктивність системи. Вартість цієї операції незначна в порівнянні з вартістю всього сервера, а результат найчастіше не виправдовує очікування користувача. У результаті апаратні витрати зростають, а продуктивність системи наполегливо «не бажає» збільшуватися пропорційно числу процесорів. Те, що можуть собі дозволити дорогі і складні мейнфрейми і суперкомп’ютери, не годиться для компактних багатопроцесорних серверів.

Пропускну здатність пам’яті в таких системах можна значно збільшити шляхом застосування великої багаторівневої кеш-пам’яті. При цьому кеш- пам’ять може містити як колективні, так і приватні дані.

Приватні дані – це дані, які використовуються одним процесором, в той час як колективні дані використовуються багатьма процесорами, по суті забезпечуючи обмін між ними. Якщо кешуються колективні дані, то вони реплікуються і можуть міститися в декількох кешах. Крім скорочення затримки доступу і необхідної смуги пропускання, така реплікація даних сприяє також загальному скороченню кількості обмінів.

Однак кешування розподілених даних викликає нову проблему – когерентність кеш-пам’яті. Ця проблема виникає через те, що значення елемента даних в пам’яті,

що використовується двома різними процесорами, є доступним цим процесорам тільки через їх індивідуальні кеші.

Для побудови більш потужних систем необхідні інші підходи. Одним з них є поділ пам’яті на незалежні модулі та забезпечення можливості доступу різних процесорів до різних модулів одночасно. Можливих рішень може бути багато, зокрема, використання матричного комутатора. Процесори і модулі пам’яті зв’язуються так, як показано на рис. 13.5. Три одночасно включених елемента, що дозволяють з'єднуватися наступним парам «центральний процесор – модуль пам'яті»: (C2, M1), (C3, M2) і (C4, M4).

На перетині ліній розташовуються елементарні точкові перемикачі, що дозволяють або забороняють передачу інформації між процесорами і модулями пам’яті. Безумовною перевагою такої організації є можливість одночасної роботи процесорів з різними модулями пам’яті. Природно, шо в ситуації, коли два процесори захочуть працювати з одним модулем пам’яті, один з них буде заблокований.

Недоліком матричних комутаторів є великий обсяг необхідного обладнання, оскільки для зв’язку N процесорів з N модулями пам’яті потрібно N2 елементарних перемикачів. У багатьох випадках це є занадто дорогим рішенням, що змушує розробників шукати інші шляхи.

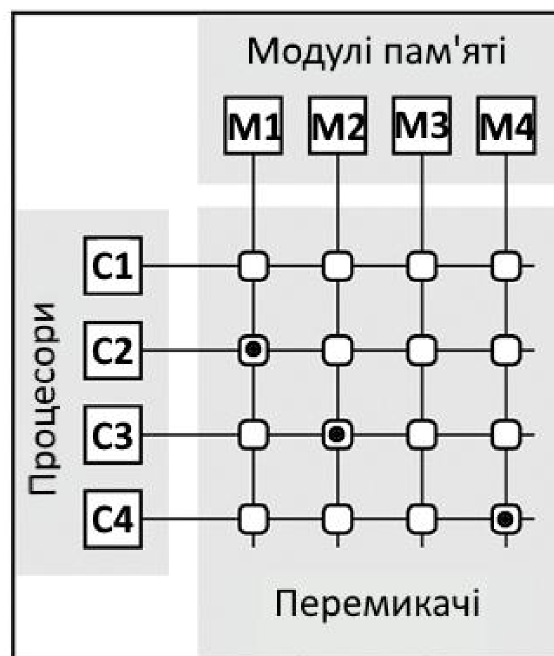

Рисунок 13.5 – Мультипроцесорна система з матричним комутатором

13.4.2 Мультипроцесорні ОС

Тепер перейдемо від апаратного забезпечення мультипроцесорів до їх програмного забезпечення, зокрема до мультипроцесорних операційних систем. Можуть бути різні підходи до їх організації, три з яких будуть розглянуті далі. Слід зауважити, що всі ці підходи можна в рівній мірі застосувати як до багатоядерних систем, так і до систем, складених з окремих центральних процесорів.

Використання власної ОС для кожного центрального процесора. Найпростіший з можливих способів організації мультипроцесорної ОС полягає в статичному розподілі пам’яті на кілька розділів за кількістю наявних центральних процесорів і виділення кожному центральному процесору власної пам’яті і своєї копії ОС (рис. 13.6) [9].

Рисунок 13.6 – Поділ пам’яті між чотирма CPU із загальною копією коду ОС

Фактично N центральних процесорів працюють як N незалежних комп’ютерів. Очевидний варіант оптимізації – це дозволити всім CPU спільно використовувати код ОС і зберігати тільки індивідуальні копії даних, Квадратики, помічені словом Data, означають персональні дані ОС для кожного CPU.

Така схема краще, ніж N незалежних комп’ютерів, так як вона дозволяє всім машинам спільно використовувати набір дисків та інших пристроїв введення-виведення, а також забезпечує гнучке спільне використання пам’яті.

Наприклад, якщо потрібно запустити велику програму, одному з CPU може бути виділена велика порція пам’яті на час виконання цієї програми. Крім того, процеси можуть ефективно спілкуватися один з одним, якщо одному процесу буде дозволено писати дані в пам’ять, а інший процес буде їх зчитувати в цьому місці.

З точки зору операційних систем наявність ОС у кожного CPU є вкрай примітивним підходом. Слід зазначити три основні недоліка даної схеми мультипроцесорної ОС:

1. Коли процес звертається до системного виклику, то системний виклик перехоплюється і обробляється його власним CPU за допомогою структур даних в таблицях ОС.

2. Оскільки у кожної ОС є свої власні таблиці, у неї є також і свій набір процесів, які вона сама планує. Спільного використання процесів немає. Якщо користувач реєструється на CPU 1, то всі його процеси працюють на CPU 1. У результаті може статися так, що CPU 1 виявиться завантажений роботою, тоді як CPU 2 буде простоювати.

3. Спільного використання сторінок також немає. Може трапитися так, що у CPU 2 багато вільних сторінок, в той час як CPU 1 буде постійно займатися свопінгом. І немає ніякого способу зайняти вільні сторінки в сусіднього CPU, так як виділення пам’яті статично фіксоване.

З причини, наведених вище недоліків, така модель тепер використовується нечасто, хоча вона застосовувалася на зорі епохи мультипроцесорів, коли ставили за мету просто перенести існуючі ОС на будь-який новий мультипроцесор якомога швидше.

Мультипроцесори, що працюють за схемою «головний-підлеглий». Друга модель показана на рис. 13.7 [9]. Тут використовується одна копія операційної системи, а її таблиці існують виключно для центрального процесора

1. Всі системні виклики переадресовуються для подальшої обробки центральному процесору 1.

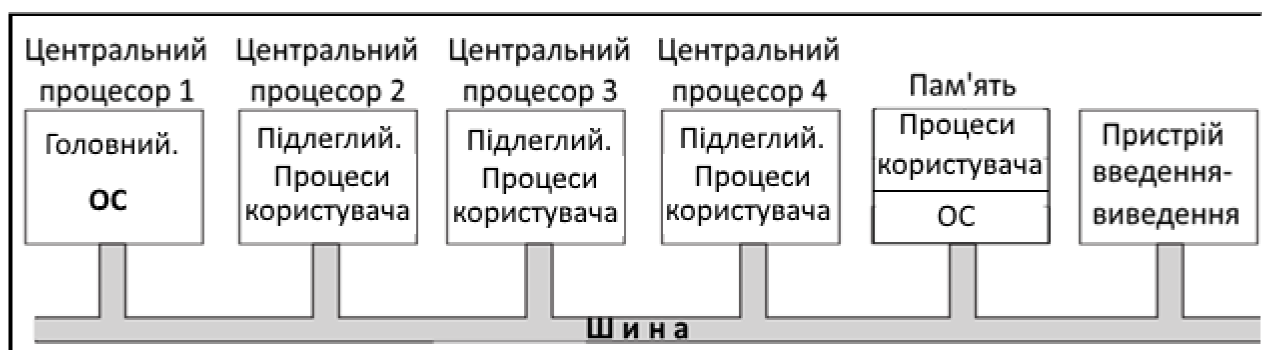

Рисунок 13.7 – Мультипроцесорна модель «головний-підлеглий»

Цей центральний процесор, якщо йому на це вистачає часу, може також запускати процеси користувача. Ця модель називається «головний-підлеглий», тому що центральний процесор 1 є головним, а всі інші – підлеглими.

Модель мультипроцесора «головний-підлеглий» дозволяє вирішити більшість проблем першої моделі.

У цій моделі використовується єдина структура даних (наприклад, один загальний список або набір пріоритетних списків), що враховує готові процеси. Коли CPU переходить в стан простою, він запитує в ОС процес, який можна обробляти, і при наявності готових процесів ОС призначає цьому CPU процес. Тому при такій організації: ніколи не може статися так, що один CPU буде простоювати, в той час як інший CPU перевантажений. Сторінки пам’яті можуть динамічно надаватися всім процесам.

Недолік такої моделі: при великій кількості центральних процесорів CPU-господар може стати вузьким місцем системи. Адже йому доводиться обробляти системні переривання від усіх CPU. Наприклад, якщо обробка системного переривання займає 10% часу, тоді 10 центральних процесорів дадуть йому граничне навантаження, а при 20 процесорах головний процесор вже не буде встигати їх обробляти, і система почне простоювати. Отже, така модель проста і працездатна для невеликої кількісті мультипроцесорів, але з великою кількістю процесорів вона працювати ефективно не може.

Симетричні мультипроцесори. Третя модель, що представляє собою симетричні мультипроцесори (SMP), дозволяє усунути перекіс попередньої моделі. Як і в попередній схемі, в пам’яті знаходиться всього одна копія ОС, але виконувати її може будь-який CPU. При системному виклику на CPU, яка звернулася до системи з системним викликом, відбувається переривання з переходом в режим ядра і обробкою системного виклику (рис. 13.8) [9].

Ця модель забезпечує динамічний баланс процесів і пам’яті, оскільки в ній є всього один набір таблиць ОС. Вона також дозволяє уникнути простою системи, пов’язаного з перевантаженням ведучого (головного) CPU, так як в ній немає ведучого CPU.

Рисунок 13.8 – Модель симетричного мультипроцесора

І все ж ця модель має власні проблеми. Зокрема, якщо два CPU одночасно беруть один і той же процес для запуску або запитують одну і ту ж вільну сторінку пам'яті, може статися катастрофа. Найпростіший спосіб вирішення подібних проблем полягає в зв'язуванні мьютекса (тобто блокування) з ОС, в результаті чого вся система перетворюється в одну велику критичну область. Коли CPU хоче виконувати код ОС, він повинен спочатку отримати мьютекс. Якщо мьютекс заблокований, CPU змушений чекати. Таким чином, будь-який CPU може виконати код ОС, але в кожен момент часу тільки один з них буде робити це.

13.4.3 Планування роботи мультипроцесора

На мультипроцесорі планування двовимірне. Планувальник повинен вирішити, який процес і на якому CPU запустити. Цей додатковий вимір істотно ускладнює планування на мультипроцесорах.

Інший ускладнюючий фактор полягає в тому, що в деяких системах всі процеси є незалежними, тоді як в інших системах вони формують групи залежних процесів.

Прикладом першої ситуації є система розподілу часу, в якій незалежні користувачі запускають незалежні процеси. Ці процеси не пов'язані один з одним, і планування кожного з них не залежить від інших процесів.

Приклад другий ситуації часто зустрічається в середовищі розробки великих програм. Великі системи, як правило, складаються з деякої кількості заголовків файлів, що містять макроси, визначення типів і оголошення змінних, які використовуються у файлах програми.

Розподіл часу. Розглянемо спочатку випадок планування незалежних процесів. Найпростіший алгоритм планування незалежних процесів (або потоків) полягає в підтримці єдиної структури даних для готових процесів, можливо, просто списку, але швидше за все множини списків для процесів з різними пріоритетами (рис. 13.9, а) [9]. Тут всі 16 CPU в даний момент зайняті, а 14 процесів з різними пріоритетами очікують запуску.

Першим закінчує роботу (або його процес блокується) CPU 4. При цьому він блокує чергу планування і вибирає з неї процес з найвищим пріоритетом, тобто процес A (рис. 13.9, б). Потім звільняється CPU 12 і вибирає процес B (рис. 13.9, в). Поки ці процеси незалежні, подібне планування являє собою розумний вибір.

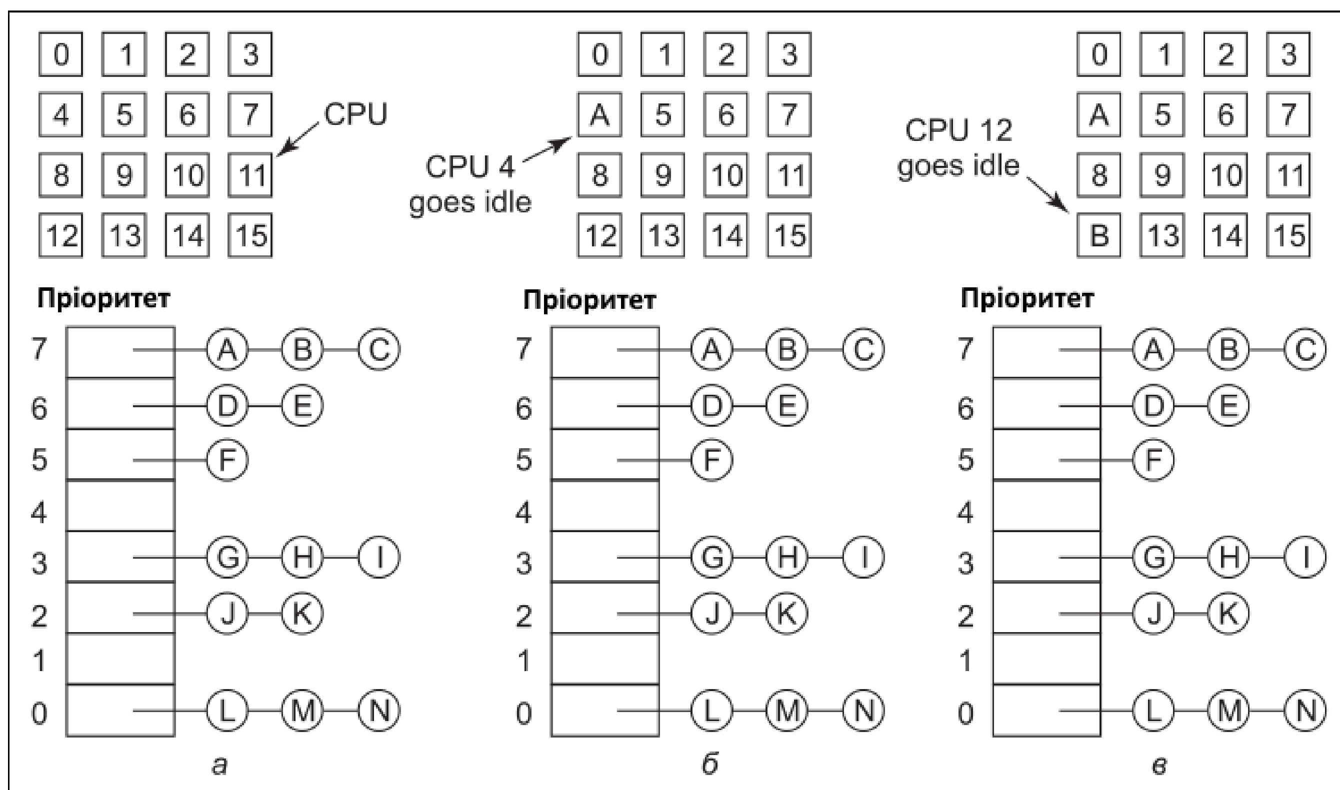

Рисунок 13.9 – Використання єдиної структури даних для планування мультипроцесора

Наявність єдиної структури даних планування, яка використовується всіма процесорами, забезпечує процесору режим розподілу часу подібно до того, як це виконується на однопроцесорній системі. Крім того, така організація дозволяє

автоматично балансувати навантаження, тобто вона виключає ситуацію, при якій один процесор простоює, в той час як інші процесори перевантажені.

Відзначимо два недоліки такої схеми планування:

- потенційне зростання конкуренції за структуру даних планування в міру збільшення числа CPU;
- звичайні накладні витрати на виконання перемикання контексту, коли процес блокується, чекаючи виконання операції введення-виведення.

Перемикання контексту також може статися, коли закінчується квант часу процесу. Припустимо, що процес утримує спін-блокування. Спін-блокування є взаємним блокуванням пристрою, яке може мати тільки два значення: «заблоковано» і «розблоковано». Тому інші CPU, які чекають звільнення блокування, просто втрачають час в циклах очікування, поки цей процес не буде запущений знову і не відпустить блокування.

На однопроцесорних системах спін-блокування практично не застосовується. Тому, якщо процес, що утримує мьютекс, блокується і запускається інший процес, то при спробі отримати мьютекс другий процес буде тут же заблокований, і багато часу втрачено не буде.

Щоб вирішити дану проблему, в деяких системах застосовується розумне планування (smart scheduling), в якому процес, захоплюючий спин-блокування, встановлює прапор, який демонструє, що він в даний момент володіє спін-блокуванням. Коли процес звільняє блокування, він також очищає і прапор. Таким чином, планувальник не зупиняє процес, що утримує спін-блокування, а, навпаки, дає йому ще трохи часу, щоб той завершив виконання критичної області та відпустив мьютекс.

Спільне використання простору. Інший підхід до планування мультипроцесорів може бути використаний, якщо процеси пов’язані один з одним будь-яким способом. Будемо називати плановані об’єкти потоками, але все сказане тут справедливо і для процесів. Планування декількох потоків на декількох CPU називається спільним використанням простору або розподілом простору.

Найпростіший алгоритм розподілу простору працює наступним чином. Припустимо, що відразу створюється ціла група пов’язаних потоків. У момент їх створення планувальник перевіряє, чи є вільні CPU за кількістю створюваних потоків.

Якщо вільних CPU досить, кожному потоку виділяється власний CPU, тобто CPU працює в однозадачному режимі, і всі потоки запускаються. Якщо CPU недостатньо, то жоден з потоків не запускається, поки не звільниться достатня кількість CPU. Кожен потік виконується на своєму CPU аж до завершення, після чого все CPU повертаються в пул вільних CPU.

Якщо потік виявляється заблокованим операцією введення-виведення, він продовжує утримувати CPU, який простоює до тих пір, поки потік не зможе продовжувати свою роботу. При появі наступного пакета потоків застосовується той же алгоритм.

У будь-який момент часу декілька CPU статично розділяється на кілька підмножин, на кожному з яких виконуються потоки одного процесу. На рис. 13.10 показані підмножини з 6, 8 і 16 CPU, і 2 CPU залишилися не включеними в підмножини [9]. Згодом, у міру завершення роботи одних процесів і появи нових процесів, кількість і розміри груп CPU змінюються.

У цій простій моделі розбиття CPU на групи процес просто запитує певну кількість CPU і або відразу отримує їх, або чекає, поки вони не звільняться.

Один із способів активного управління ступенем розпаралелювання процесів полягає в наявності центрального сервера, який веде облік працюючих і охочих працювати процесів, а також мінімального і максимального кількості потрібних для них CPU. Періодично кожен CPU опитує центральний сервер, щоб дізнатися, скільки CPU він може використовувати. Потім він збільшує або зменшує кількість процесів або потоків, намагаючись домогтися відповідності числа доступних CPU.

Рисунок 13.10 – Набір з 32 CPU, розділений на чотири групи

Бригадне планування. Явною перевагою спільного використання простору є виняток багатозадачності, що знижує накладні витрати по переключенню контексту. Однак її недолік полягає у втрати часу при блокуванні CPU.

Щоб зрозуміти, які можливі проблеми при незалежному плануванні потоків процесу (або процесів завдання), розглянемо систему з потоками A0 і A1, що належать процесу A, і потоками B0 і B1, що належать процесу B. Потоки A0 і B0 працюють в режимі розподілу часу на CPU 0, а потоки A1 і B1 – на CPU 1. Потокам A0 і A1 потрібно часто обмінюватися інформацією.

Спілкування потоків виглядає наступним чином. Потік A0 посилає потоку A1 повідомлення, після чого потік A1 відправляє потоку A0 відповідь і т. д. Припустимо, що потоки A0 і B1 почали виконуватися першими, як показано на рис. 13.11 [9].

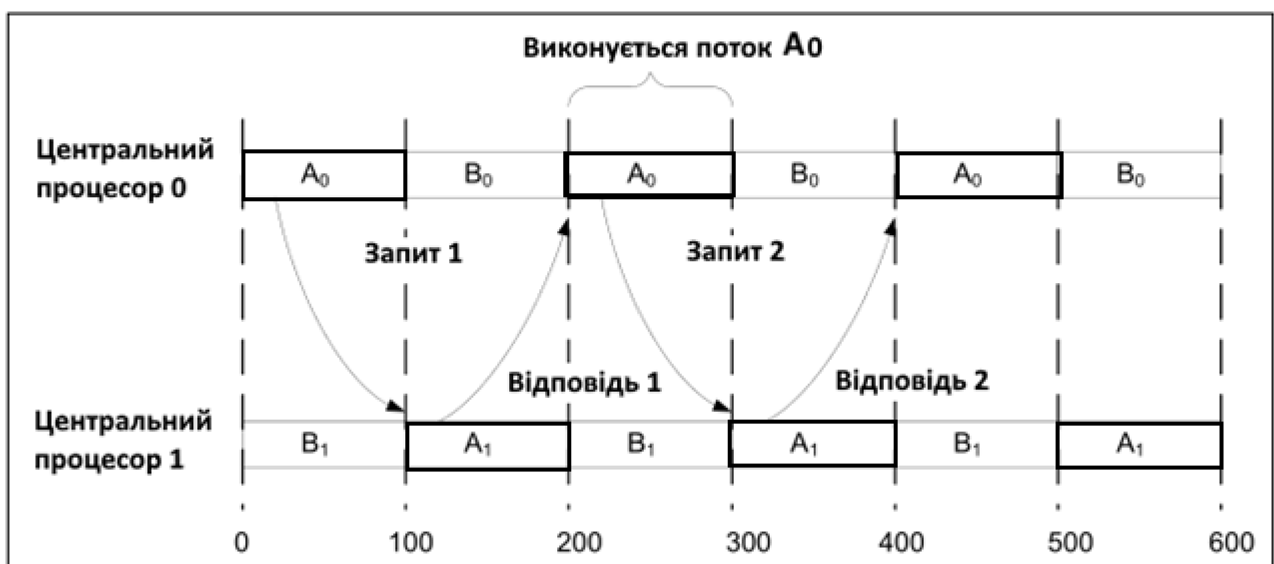

Рисунок 13.11 Спілкування двох потоків, що належать процесу A

Рішенням даної проблеми є так зване бригадне планування, що представляє собою розвиток ідеї спільного планування. Бригадне планування складається з трьох частин:

1. Групи пов’язаних потоків плануються як одне ціле – бригада.

2. Всі члени бригади запускаються одночасно на різних центральних процесорах, що працюють в режимі розподілу часу.

3. У всіх членів бригади кванти часу починаються і закінчуються одночасно. Бригадне планування працює завдяки синхронності роботи всіх CPU. Це означає, що час розділяється на дискретні кванти, як було показано на рис. 13.11. На початку

кожного нового кванта всі CPU переплановуються заново, і на кожному CPU запускається новий потік. На початку наступного кванта знову приймається рішення про планування. В середині кванта планування не виконується. Якщо який-небудь потік блокується, його CPU простоює до кінця кванта часу. Приклад роботи бригадного планування наведено на рис. 13.12.

		Центральний процесор					
		0	1	2	3	4	5
Часовий інтервал	0	A_0	A_1	A_2	A_3	A_4	A_5
	1	B_0	B_1	B_2	C_0	C_1	C_2
	2	D_0	D_1	D_2	D_3	D_4	E_0
	3	E_1	E_2	E_3	E_4	E_5	E_6
	4	A_0	A_1	A_2	A_3	A_4	A_5
	5	B_0	B_1	B_2	C_0	C_1	C_2
	6	D_0	D_1	D_2	D_3	D_4	E_0
	7	E_1	E_2	E_3	E_4	E_5	E_6

Рисунок 13.12 – Бригадне планування

Тут показаний мультипроцесор з шістьма CPU, на яких працюють п’ять процесів, від A до E, із загальним числом потоків, рівним 24. Протягом часового інтервалу 0 потоки від A0 до A5 плануються і виконуються. Під час інтервалу 1 плануються і виконуються B0-B2 і C0-C2. Протягом часового інтервалу 2 плануються і виконуються п’ять потоків процесу D і потік E0. Решта шість потоків процесу E працюють під час інтервалу 3. Потім цикл повторюється так, що часовий інтервал 4 повторює інтервал 0 і т. д.

Ідея бригадного планування полягає в тому, щоб всі потоки процесу працювали по можливості разом, так, якщо один із потоків посилає повідомлення іншому потоку, то другий потік отримує повідомлення практично миттєво і може так само швидко на нього відповісти. Оскільки всі потоки процесу A працюють разом протягом одного кванта часу, вони можуть відправляти і приймати велику кількість повідомлень за один квант часу, усуваючи, таким чином, проблему, зображену на рис. 13.11.