6.1 КОНЦЕПЦІЇ ПОТОКУ

У системах, в яких відсутнє поняття потоку, виникають проблеми при організації паралельних обчислень у рамках процесу. А така необхідність може виникати. Свого часу впровадження ідеї мультипрограмування дозволило підвищити пропускну спроможність комп'ютерних систем, тобто зменшити середній час очікування результатів роботи процесів. Але всякий розподіл ресурсів тільки уповільнює роботу одного з учасників за рахунок додаткових витрат часу на очікування звільнення ресурсу. Проте додаток, що виконується в рамках одного процесу, може мати внутрішній паралелізм, який в принципі міг би дозволити прискорити його роботу.

Потоки виникли в операційних системах як засіб розпаралелювання обчислень. Звичайно, задача розпаралелювання обчислень у рамках одного додатку може бути вирішене і традиційними способами, як це ми бачили на наведеному вище прикладі.

По-перше, прикладний програміст може взяти на себе складну задачу організації паралелізму, виділивши в додатку деяку підпрограму-диспетчер, яка періодично передає управління тієї ж або іншій гілки обчислень. При цьому програма виходить логічно дуже заплутаною, з численними передачами управління, що істотно ускладнює її відлагодження і модифікацію.

По-друге, використання стандартних засобів ОС для створення процесів не дозволяє врахувати той факт, що ці процеси розв'язують єдину задачу, тобто мають багато спільного між собою. Вони можуть працювати з одними і тими ж даними, використовувати один і той же кодовий сегмент, наділятися одними і тими ж правами доступу до ресурсів обчислювальної системи. Так, якщо в прикладі з сервером баз даних створювати окремі процеси для кожного запиту, що поступає з мережі, то усі процеси виконуватимуть один і той же програмний код і виконуватимуть пошук в записах, загальних для всіх процесів файлів даних. А операційна система при такому підході розглядатиме ці процеси нарівні з усіма іншими процесами і за допомогою універсальних механізмів забезпечуватиме їх ізоляцію один від одного.

У даному випадку всі ці досить громіздкі механізми використовуються явно не за призначенням, виконуючи не лише даремну, але і шкідливу роботу, що утрудняє

обмін даними між різними частинами додатку. Крім того, на створення кожного процесу ОС витрачає певні системні ресурси, які в даному випадку невиправдано дублюються, – кожному процесу виділяються власний віртуальний адресний простір, фізична пам'ять, за ним закріплюються пристрої введення-виведення тощо.

Якщо, наприклад, у програмі передбачено звернення до зовнішнього пристрою, то на час цієї операції можна не блокувати виконання всього процесу, а продовжити обчислення по іншій гілці програми. Паралельне виконання декількох робіт у рамках одного інтерактивного додатку підвищує ефективність роботи користувача. Так, при роботі з текстовим редактором бажано мати можливість поєднувати набір нового тексту з такими тривалими за часом операціями, як переформатування значної частини тексту, друк документу або його збереження на локальному або віддаленому диску.

Ще одним прикладом необхідності розпаралелювання є мережевий сервер баз даних. У цьому випадку паралелізм бажаний як для обслуговування різних запитів до бази даних, так і для швидшого виконання окремого запиту за рахунок одночасного перегляду різних записів бази даних.

Розглянемо простий приклад. Нехай у нас є така програма на псевдомові програмування.

```
Ввести масив A
Ввести масив B
Ввести масив C
A = A + B; C = A + C
Вивести масив C
```

При виконанні такої програми в рамках одного процесу цей процес чотири рази блокуватиметься, чекаючи закінчення операцій введення-виведення. Але наш алгоритм має внутрішній паралелізм. Обчислення суми масивів A + B можна було б виконувати паралельно з очікуванням закінчення операції введення масиву C. Таке поєднання операцій за часом можна було б реалізувати, використовуючи два взаємодіючі процеси. Для простоти вважатимемо, що засобом комунікації між ними служить пам'ять, що розділяється. Тоді наші процеси можуть виглядати таким чином.

Процес 1	Процес 2
Ввести масив A	Очікування введення масивів A і B
Очікування закінчення операції введення	
Ввести масив B	
Очікування закінчення операції введення	
Ввести масив C	
Очікування закінчення операції введення	A = A + B
C = A + C	
Вивести масив C	
Очікування закінчення операції виведення	

Здавалося б, ми запропонували конкретний спосіб прискорення розв'язання задачі. Проте, насправді, справа йде не так просто. Другий процес має бути створений, обидва процеси повинні повідомити операційну систему, що їм потрібна пам'ять, яку вони могли б розділити з іншим процесом, і, нарешті, не можна забувати про перемикання контексту. Тому реальна поведінка процесів виглядатиме приблизно так.

Процес 1	Процес 2
Створити процес 2	
Перемикання контексту	
	Виділення загальної пам'яті
	Очікування введення A і B
Перемикання контексту	
Виділення загальної пам'яті	
Ввести масив A	
Очікування закінчення операції введення	
Ввести масив B	
Очікування закінчення операції введення	
Ввести масив C	
Очікування закінчення операції введення	
Перемикання контексту	
	A = A + B
Перемикання контексту	
C = A + C	
Вивести масив	
Очікування закінчення операції виведення	

Очевидно, що ми можемо не лише не виграти в часі при розв'язанні задачі, але навіть і програти, оскільки часові втрати на створення процесу, виділення загальної пам'яті і перемикання контексту можуть перевищити виграш, отриманий за рахунок поєднання операцій.

З усього вищевикладеного витікає, що в ОС, разом з більшою одиницею роботи (процесами), потрібний інший механізм розпаралелювання обчислень, який враховував би тісні зв'язки між окремими гілками обчислень одного і того ж додатку, і вимагав для свого виконання дещо дрібніших робіт. Для цих цілей сучасні ОС пропонують механізм багатопотокової обробки (multithreading). При цьому вводиться нова одиниця роботи – потік виконання [9], а поняття «процес» значною мірою міняє сенс. Поняттю «потік» відповідає послідовний перехід процесора від однієї команди програми до іншої. ОС розподіляє процесорний час між потоками. ОС призначає процесу адресний простір і набір ресурсів, які спільно використовуються усіма його потоками.

Говорячи про процеси, ми відмічали, що ОС підтримує їх відособленість: у кожного процесу є свій віртуальний адресний простір, кожному процесу призначаються свої ресурси – файли, вікна тощо. Така відособленість потрібна для того, щоб захистити один процес від іншого, оскільки вони, спільно використовуючи усі ресурси машини, конкурують один з одним. У загальному випадку процеси належать різним користувачам, що розділяють один комп'ютер, і ОС бере на себе роль арбітра в спорах процесів за ресурси.

Проте задача, що вирішується в рамках одного процесу, може мати внутрішній паралелізм, який, в принципі, дозволяє прискорити його розв'язання. Наприклад, в ході виконання задачі відбувається звернення до зовнішнього пристрою, і на час цієї операції можна не блокувати повністю виконання процесу, а продовжити обчислення по іншій «гілці» процесу. Таким чином, для реалізації «мультизадачності» в її істинному тлумаченні необхідно теж ввести відповідну суть. Такою суттю і стали так звані процеси «легкої ваги» (полегшені процеси), або міні-процеси, або, як їх тепер називають, – потоки або треди (нитки, thread).

Створення потоків вимагає від ОС менших накладних витрат, ніж процесів. На відміну від процесів, які належать різним конкуруючим додаткам, усі потоки одного

процесу завжди належать одному додатку, тому ОС ізолює потоки в набагато меншому ступені, ніж процеси в традиційній мультипрограмній системі.

Усі потоки одного процесу використовують загальні файли, таймери, пристрої, одну і ту ж область оперативної пам'яті, один і той же адресний простір. Це означає, що вони розділяють одні і ті ж глобальні змінні. Оскільки кожен потік може мати доступ до будь-якої віртуальної адреси процесу, один потік може використати стек іншого потоку. Між потоками одного процесу немає повного захисту, оскільки, по-перше, це неможливо, а, по-друге, непотрібно. Щоб організувати взаємодію і обмін даними, потокам зовсім не потрібно звертатися до ОС, їм достатньо використати загальну пам'ять – один потік записує дані, а інший їх читає.

Другим аргументом на користь потоків є легкість (тобто швидкість) їх створення і ліквідації в порівнянні з «важкими» процесами. У багатьох системах створення потоків здійснюється в 10-100 разів швидше, ніж створення процесів. Ця властивість особливо згодиться, коли потрібно буде швидко і динамічно змінювати кількість потоків.

Отже, мультипрограмування ефективніше на рівні потоків, а не процесів. Кожен потік має власний лічильник команд і стек. Задача, яка оформлена у вигляді декількох потоків у рамках одного процесу, може бути виконане швидше за рахунок псевдопаралельного (чи паралельного в мультипроцесорній системі) виконання її окремих частин. Наприклад, якщо електронна таблиця була розроблена з урахуванням можливостей багатопотокової обробки, то користувач може запросити перерахунок свого робочого листа і одночасно продовжувати заповнювати таблицю. Особливо ефективно можна використати багатопоточність для виконання розподілених додатків, наприклад, багатопотоковий сервер може паралельно виконувати запити відразу декількох клієнтів.

Використання потоків пов'язане не лише з прагненням підвищити продуктивність системи за рахунок паралельних обчислень, але і з метою створення читабельніших, логічніших програм. Наприклад, в задачах типу «письменник-читач» один потік виконує запис у буфер, а інший прочитує записи з нього. Оскільки вони ділять між собою загальний буфер, то не потрібно їх робити окремими процесами. Інший приклад використання потоків – управління сигналами, такими як переривання

з клавіатури (del або break). Замість обробки сигналу переривання один потік призначається для постійного очікування сигналів. Таким чином, використання потоків може скоротити необхідність в перериваннях рівня користувача. У цих прикладах не таке важливе паралельне виконання, наскільки важлива ясність програми.

Найбільший ефект від введення багатопотокової обробки досягається в мультипроцесорних системах, в яких потоки, у тому числі і ті, що належать одному процесу, можуть виконуватися на різних процесорах дійсно паралельно (а не псевдопараллельно).