7.1 ПРИНЦИПИ ПАРАЛЕЛЬНИХ ОБЧИСЛЕНЬ

Основні питання, на які зосереджується увага розробників ОС, пов'язана з управлінням процесами і потоками.

1. **Багатозадачність:** управління певною кількістю процесів в однопроцесорній системі.

2. **Багатопроцесорність:** управління певною кількістю процесів у багатопроцесорній системі.

3. **Розподілені обчислення:** управління певною кількістю процесів, що виконуються в розподіленій обчислювальній системі з множиною комп'ютерів (кластери).

Розглянемо в контексті багатопроцесорності і багатозадачності проблему паралельності виконання процесів. Основною вимогою підтримки паралельних процесів є можливість забезпечення взаємовиключення, тобто можливість забезпечити роботу тільки одного процесу з призупиненням виконання інших.

У однопроцесорній багатозадачній системі процеси виконуються по черзі для створення ілюзії одночасного виконання. Попри те, що при цьому не досягається реальна паралельна робота процесів і, більше того, є певні накладні витрати, пов'язані з перемиканням між процесами, таке виконання, що чергується, забезпечує чималі вигоди з точки зору ефективності і структуризації програм.

У багатопроцесорних системах можливо не лише чергування процесів, але їх перекриття. Перекриття і чергування процесів є принципово різними режимами роботи, але їх можна розглядати як приклади паралельних обчислень, які породжують однакові проблеми.

7.1.1 Участь операційної системи

За наявності паралельних обчислень перелічимо такі питання, які виникають при створенні та управлінні ОС.

1. ОС повинна відстежувати різні активні процеси. Це виконується за допомогою блоків управління процесами.

2. ОС повинна розподіляти і звільняти різні ресурси для кожного активного процесу, а саме:

- процесорний час, це функція планування;
- пам'ять, більшість ОС використовують схему віртуальної пам'яті;
- файли системи;
- пристрої введення-виведення.

3. ОС повинна захищати дані і ресурси кожного процесу від ненавмисного впливу інших процесів.

4. Результат роботи процесу не повинен залежати від швидкості його виконання стосовно інших процесів, які виконуються паралельно.

7.1.2 Взаємодія процесів

При необхідності використати один і той же ресурс паралельні процеси вступають в конфлікт один з одним. Кожному процесу по можливості надаються свої додаткові ресурси. Проте, для вирішення деяких задач процеси можуть об'єднувати свої зусилля. Зокрема, якщо два процеси бажають отримати доступ до одного ресурсу, то ОС виділить цей ресурс одному з процесів, тоді як другий процес змушений чекати на завершення роботи з ресурсом першого. Таким чином, швидкість виконання процесу, якому відмовлено в негайному доступі до ресурсу, зменшується.

Представимо таку ситуацію, що при банківській організації системи для обслуговування клієнтів виділяють окремий потік для кожного клієнта. Припустимо, що додавання даних на вклад клієнта зводиться до збільшення глобальної змінної Amount (Сума).

Розглянемо випадок, коли два клієнти А і В спільно користуються одним і тим же рахунком. Нехай на рахунку було 100 грн. Клієнт А хоче додати до рахунку 3 грн, а клієнт В – 5 грн. Потік А відповідає клієнтові А, а В – клієнтові В.

Спочатку розглянемо ситуацію, коли обчислювальна система має один ЦП. Передбачається, що ОС використовує витіснячий алгоритм планування. У цьому випадку основа проблеми полягає в можливості передачі ЦП від одного потоку до іншого до завершення їм операції із записом у змінну Amount. На рис. 7.1 представлені три можливих варіанти діаграми виконання потоків.

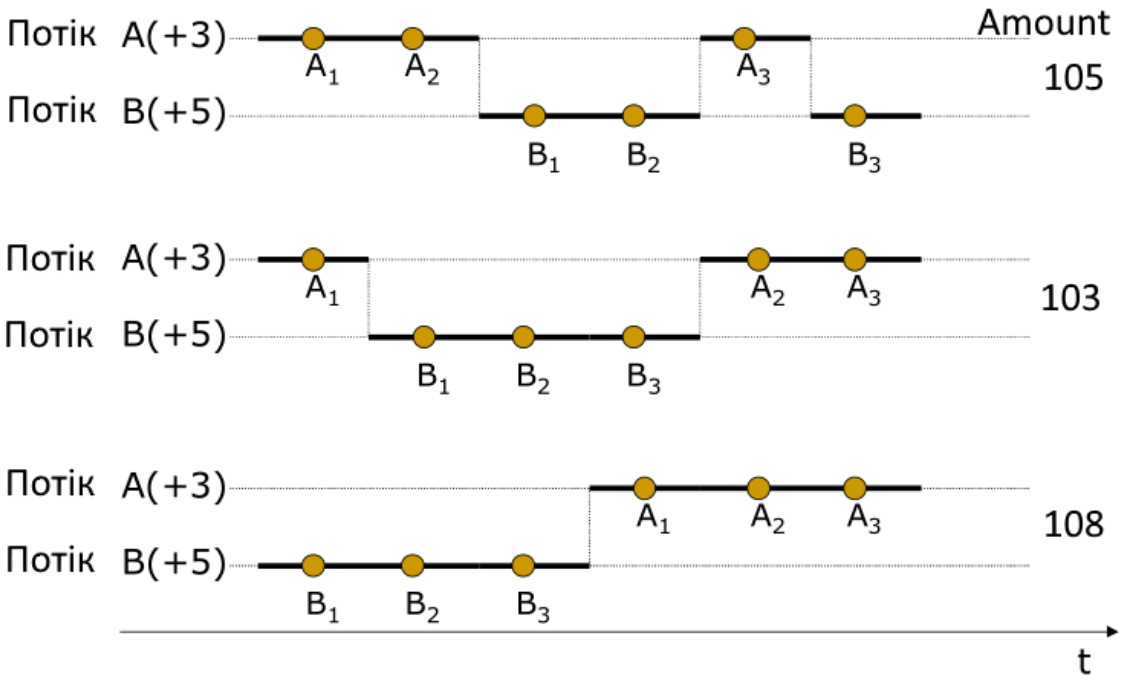

Рисунок 7.1 – Варіанти діаграми виконання потоків на однопроцесорній системі

Який саме варіант реалізується при конкретному запуску програми, залежить від взаємних швидкостей потоків і моментів передачі ЦП від одного потоку до іншого. Відзначимо, що в більшості випадків потік не може вплинути на дані фактори.

У разі багатопроцесорної системи також можливе виконання різних діаграм потоків (рис. 7.2).

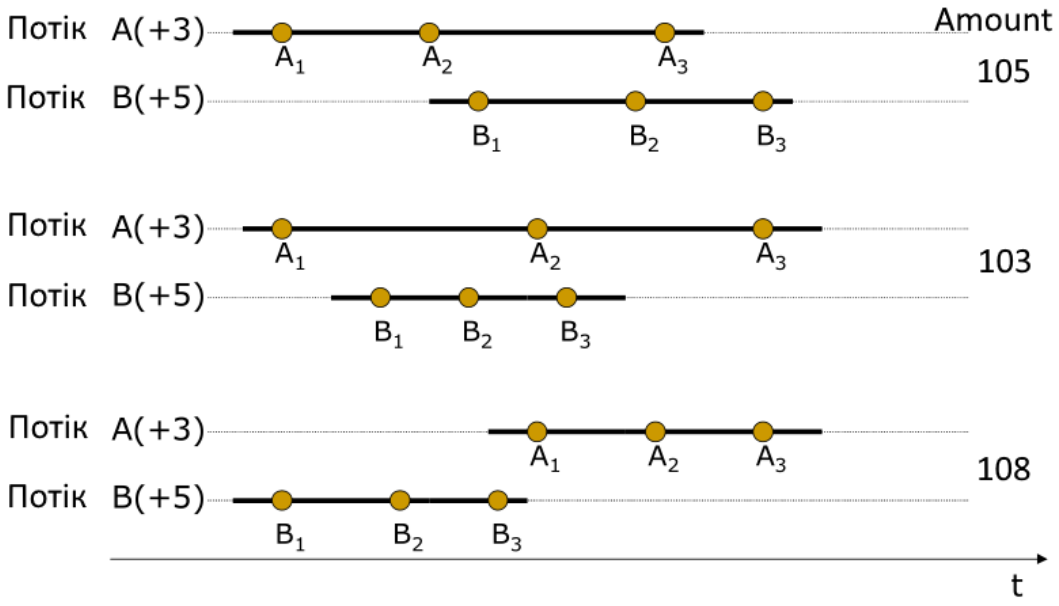

Рисунок 7.2 – Варіанти діаграми виконання потоків на багатопроцесорній системі

На даній діаграмі ми припускаємо, що потоки не витіснялися планувальником. У цьому випадку результат виконання операції залежить від відносних швидкостей потоків. Потік ніяк не може передбачити або вплинути на свою відносну швидкість: вона залежить від стану обчислювальної системи (наприклад, знаходяться його код або дані в кеш ЦП чи ні) і ОС та процесу- власника потоку (наприклад, частина сторінок, використовуваних потоком, може знаходитися на жорсткому диску).

Таким чином, результат обчислень у розглянутому прикладі не однозначний – все залежить від умов виконання потоків, і різні запуски програми можуть привести до різних результатів.

Такі ситуації, в яких два (і більше) потоків (процесів) прочитують або записують дані одночасно, і кінцевий результат залежить від того, який з них був першим, тобто від співвідношення швидкостей потоків, називають станом перегонів, або змагань (race condition). Спроби рішення подібних проблем викликали необхідність синхронізації процесів. Тобто, основне призначення синхронізації – це необхідність забезпечення захисту даних від впливу інших потоків.

У разі конкуренції процесів виникають три проблеми.

Перша – необхідність взаємних виключень. Припустимо, що два або декілька процесів вимагають доступу до одного ресурсу (введення-виведення). При виконанні кожен процес посилає команди в пристрій введення-виведення, отримує інформацію про його стан, посилає і/або отримує дані. Будемо говорити про такий ресурс як про критичний ресурс, а про частину програми, яка його використовує, – як про критичний розділ (критична секція) програми.

Критичною секцією називається послідовність операторів, які мають доступ до об'єкта, що розділяється. Тобто, критична секція – це фрагмент коду потоку, який працює з ресурсом, що розділяється між декількома потоками. Тому говорять про зайняття і звільнення одним потоком критичної секції, загальної для декількох потоків (рис. 7.3) [9].

Украй важливо, щоб в критичній секції, пов'язаній з цим ресурсом, у будь- який момент часу міг знаходитися тільки один процес. Цей прийом називають взаємним виключенням або блокуванням.

У разі блокування перевіряють, чи не було воно вже зроблено іншими процесами (потоками), а якщо це так, то цей процес переходить в очікування, інакше він здійснює блокування стану і входить в критичну секцію. Після виходу з критичної секції процес знімає блокування. Так реалізується взаємне виключення, звідси походить ще одна назва блокування – м'ютекс (mutex, скорочення від mutual exclusion).

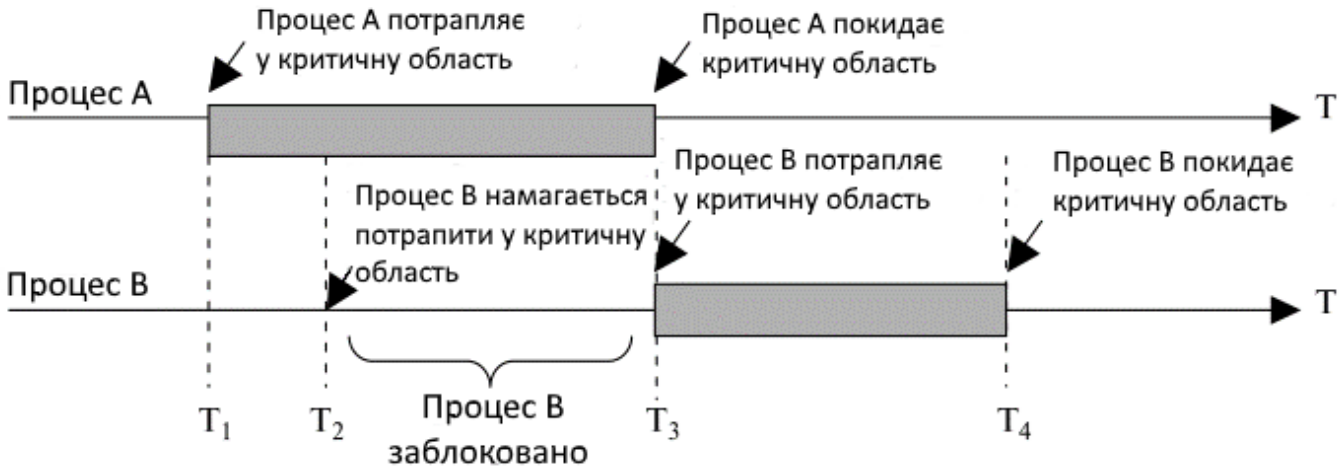

Рисунок 7.3 – Критична секція

У загальному випадку **м'ютексом** називають примітив синхронізації, який не допускає виконання деякого фрагмента коду більше як одним процесом. М’ютекс – це змінна, яка може знаходитися в одному з двох станів: блокованому (0) або неблокованому (≠ 0). М’ютекс можна розглядати як спрощений семафор, який ми розглядатимемо пізніше. Звідси витікає, що вирішення проблеми змагань (race condition) є перетворенням коду в атомарну операцію.

При реалізації взаємних виключень виникають додатково дві проблеми. Одна з них – взаємне блокування. Розглянемо два процеси P1 і P2 і два ресурси R1 і R2. Припустимо, що кожному процесу для виконання деяких функцій потрібний доступ до обох ресурсів. При цьому можлива ситуація: ОС виділяє ресурс R1 процесу P2, а ресурс R2 – процесу P1. Процеси будуть взаємно заблоковані.

Остання проблема – голодування. Припустимо, що ми маємо три процеси P1, P2, P3, кожному з яких потрібний доступ до ресурсу R. Припустимо, що P1 утримує ресурс R, а P2 і P3 призупинені в очікуванні звільнення ресурсу. Після виходу P1 з критичного розділу доступ до ресурсу отримує P2 або P3. Припустимо, що ОС надала

доступ до ресурсу R процесу P3. Поки він працює з ресурсом, доступ до ресурсу знову потрібен процесу P1. У результаті може статися, що ОС знову надасть ресурс R процесу P1. Потім доступ знову може знадобитися P3. Таким чином, можлива ситуація, коли процес P2 ніколи не отримає ресурс R.

Управління конкуренцією неминуче призводить до участі ОС у цьому процесі, оскільки саме вона розподіляє ресурси. Процесам також потрібна можливість запитувати взаємовиключення, таке, як блокування ресурсу перед його використанням. Будь-яке розв'язання цього питання вимагає підтримки ОС, наприклад, такої, як забезпечення блокування.

Для усунення таких ситуацій може бути використаний так званий **апарат подій**. За допомогою цього засобу можуть вирішуватися не лише проблеми взаємного виключення, але й загальніші задачі синхронізації процесів. У різних ОС апарат подій реалізується по-своєму, але в будь-якому випадку використовуються системні функції аналогічного призначення.

7.1.3 Вимоги до взаємних виключень

Будь-яка можливість забезпечення підтримки взаємних виключень повинна відповідати таким вимогам, наведеним нижче.

1. Взаємовиключення повинно здійснюватися в примусовому порядку. У будь-який момент часу з усіх процесів, що мають критичний розділ для одного і того ж ресурсу, в цьому розділі може знаходитися тільки один процес.

2. Процес, що завершує роботу в некритичному розділі, не повинен впливати на інші процеси.

3. Не повинна виникати ситуація нескінченного очікування доступу до критичного розділу, тобто не повинні з'являтися взаємоблокування і голодування.

4. Коли в критичному розділі немає жодного процесу, будь-який процес, що запросив можливість входу в нього, повинен його негайно отримати.

5. Не повинно існувати ніяких припущень про відносні швидкості процесів, що виконуються, або число процесорів, на яких вони виконуються.

6. Процес залишається в критичному розділі впродовж обмеженого часу.

Є ряд способів задоволення перерахованих умов. Одним з них є передача відповідальності за відповідність вимогам самому процесу, який повинен виконуватися паралельно. Таким чином, процес, незалежно від того, чи є він системною програмою чи додатком, повинен координувати свої дії з іншими процесами для роботи взаємовиключень без підтримки з боку ОС.