Лекція 3. Концепції потокової обробки

План лекції:

1. Визначення програмних потоків
2. Поточна обчислювальна модель
3. Конструкції паралельного програмування

1. Визначення програмних потоків

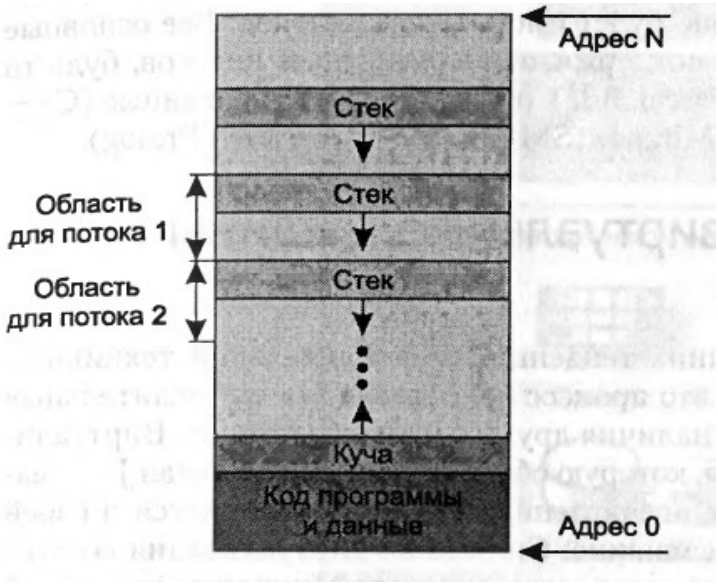
Програмний потік – це окрема послідовність взаємозв'язаних між собою команд, яка виконується незалежно від інших послідовностей команд. Кожна програма має хоча б один програмний потік – головний. Цей потік ініціалізує програму та починає виконання перших команд. Він може створювати інші потоки, які будуть виконувати різні завдання.

Кожен програмний потік підтримує свій поточний машинний стан, тобто адресу виконуваної команди, адреси та значення даних, які знаходяться в регістрах процесора або в пам'яті.

Щоб зрозуміти, яким чином реалізувати потокову обробку в своєму додатку, потрібно добре знати наступні моменти:

- структуру додатку та підходи до його розробки;
- потоковий програмний інтерфейс API;
- компілятор або середовище виконання додатку;
- цільові апаратні платформи, на яких буде виконуватися додаток.

Кожен потік повинен мати свій стековий простір. Управління стеками звичайно виконує операційна система. На мал. 3.1 наведено типовий стек багатопоточного процесу.



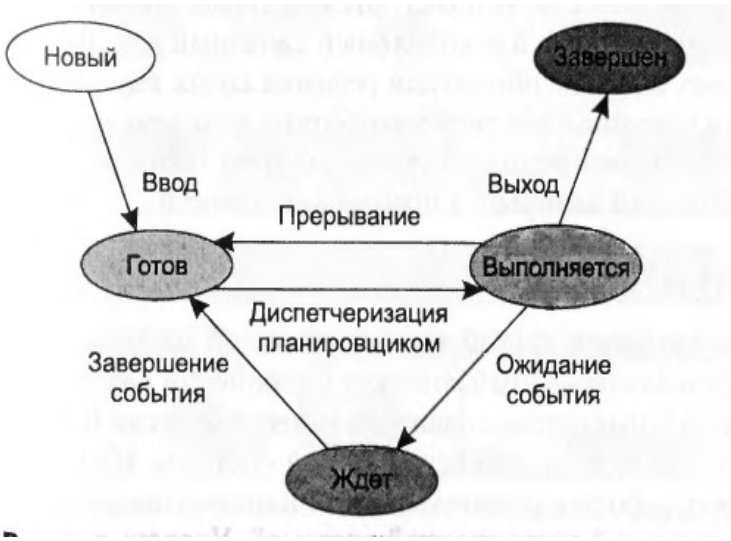
Мал. 3.1. Вигляд стеку багатопоточного процесу

Розмір стеку за замовчуванням в різних операційних системах відрізняється. Тому у деяких системах створення великої кількості потоків може значно знизити продуктивність.

Після створення потік знаходиться в одному з чотирьох станів:

- готовність;
- виконання;
- очикування;
- завершення.

На мал. 3.2. наведена діаграма станів потоку.



Мал. 3.2. Діаграма станів потоку

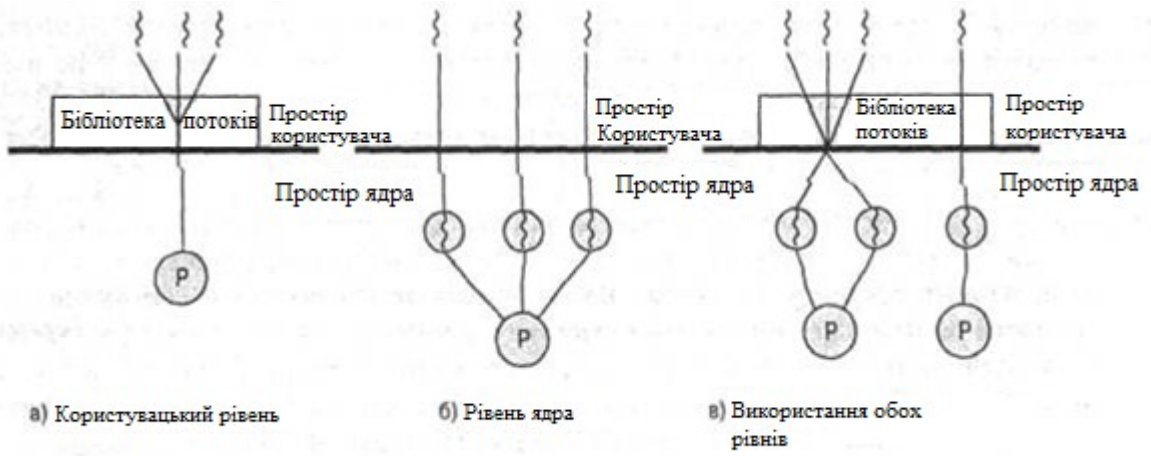
Як показано на малюнку, кожен створений потік буде знаходитися в стані готовності. Після цього, коли новий потік виконує команди він буде знаходитися в стані виконання. Якщо потік потребує якогось ресурсу, або його перериває інший потік, він переходить в стан очікування. При закінченні роботи потоку, він або завершується, або переходить в стан готовності. Після завершення програми головний та допоміжні потоки також завершуються.

Процеси та потоки представляють різні рівні виконавчого механізму системи. Для того, щоб вірно оцінити можливості поточної обробки, важливо зрозуміти вплив потоків на компоненти системи.

2. Поточна обчислювальна модель

Поточну обчислювальну модель (мал. 3.3) складають три рівня поточної обробки:

- *Потоки рівня користувача* створюються і керуються із додатків;
- *На рівні ядра* в операційній системі реалізуються більшість програмних потоків;
- *Апаратними* потоки являються для виконавчих ресурсів апаратного забезпечення.



Мал. 3.3. Поточна обчислювальна модель

Потоки виконання ядра відносяться до «легких» одиниць планування ядра. Всередині кожного процесу існує принаймні один потік виконання ядра. Якщо в рамках процесу можуть існувати кілька потоків виконання ядра, то вони спільно використовують загальну пам'ять і файл ресурсів. Якщо процес виконання планувальника операційної системи є пріоритетним, то потоки виконання ядра теж є пріоритетно багатозадачними. Потоки виконання ядра не мають власних ресурсів, за винятком стека викликів, копії регістрів процесора, включаючи лічильник команд, і локальну пам'ять потоку виконання (якщо вона є). Ядро може призначити по одному потоку виконання для кожного логічного ядра системи (оскільки кожен процесор поділяє сам себе на декілька логічних ядер, якщо він підтримує багатопоточність, або підтримує лише одне логічне ядро на кожне фізичне ядро, якщо не підтримує багатопоточність), а може виконувати свопінг заблокованих потоків виконання. Однак потоки виконання ядра вимагають набагато більше часу, ніж потрібно на свопінг настроюваних потоків виконання.

Потоки виконання іноді реалізуються в користувацькому просторі бібліотек, в цьому випадку вони називаються призначеними для користувача потоками виконання. Ядро не знає про них, так що вони управляються і плануються в користувацькому просторі. У деяких реалізаціях користувальницькі потоки виконання ґрунтуються на кількох верхніх потоках виконання ядра, щоб використовувати переваги багатопроцесорних машин (моделі M:N). У даній статті під терміном «потік виконання» за замовчуванням (без кваліфікатора «ядра» або «користувацький») мається на увазі «потік виконання ядра». Користувальницькі потоки виконання, реалізовані з допомогою віртуальних машин, називають «зеленими потоками виконання». Користувальницькі потоки виконання, як правило, можна швидко створювати, і ними легко керувати, але вони не можуть використовувати переваги багатопоточності і багатопроцесорності. Вони

можуть блокуватися, якщо всі пов'язані з ним потоки виконання ядра зайняті, навіть якщо деякі користувальницькі потоки готові до запуску.

Апаратний потік — це окрема частина обладнання, яка виконує код. Десять років тому в окремо взятому процесорі був всього один апаратний потік, а більша кількість таких потоків зустрічалася лише в комп'ютерах з декількома фізично незалежними процесорами, підключеними до окремих портів на материнській платі. Однак два нововведення, а саме багатоядерні процесори і гиперпоточность, ускладнили взаємодію між потоками і апаратною частиною.

Багатоядерний процесор — це фактично кілька процесорів на одній кремнієвій схемі. Іншими словами, якщо ви знімете кришку комп'ютера і порахуйте, скільки у вас процесорних мікросхем, то не обов'язково отримаєте число апаратних потоків. Але якщо подивитися на процесор під хорошим мікроскопом, то можна побачити два і більше процесора на одному чіпі.

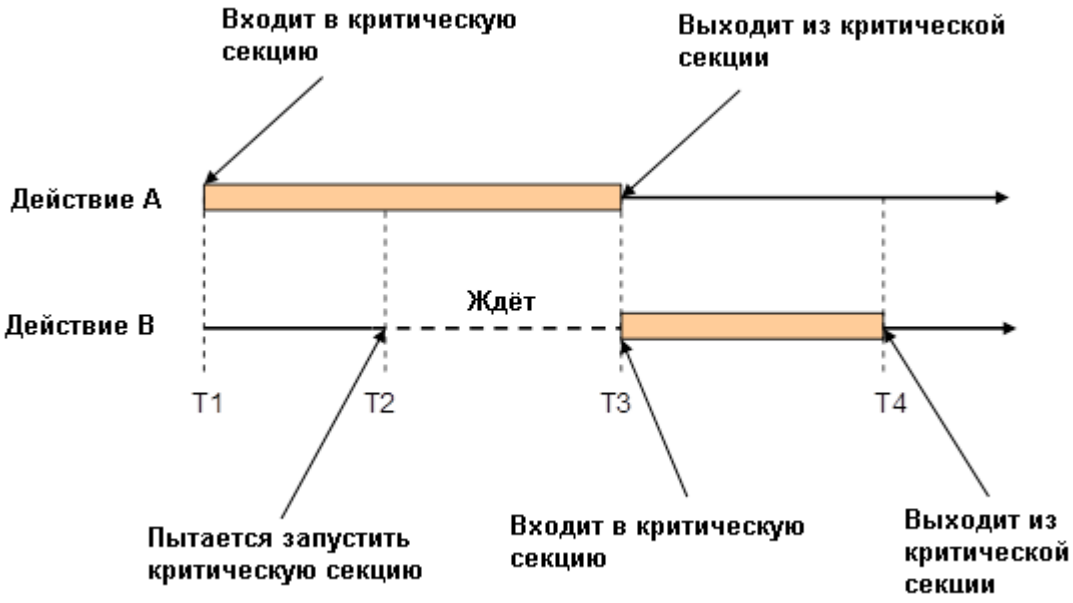
Гіперпоточность, також іменована одночасна багатопоточність (SMT), все ускладнює ще більше. На багатопотоковому ядрі певні елементи процесора дублюються (іноді копій ще більше, однак найчастіше використовується саме дублювання). В такому випадку лише одна частина процесора може проводити, скажімо, ділення з плаваючою точкою, але при цьому існують два комплекти регістрів і логіки для набору команд. Серед регістрів є лічильник команд, стежить за виконанням і поточним робочим станом коду, так що два набору регістрів дають можливість одночасно запускати два фрагменти коду на одному ядрі. Іншими словами, гіперпоточность дозволяє одному ядру мати два апаратних потока. Два контексти виконання змушені спільно використовувати ресурси — вони не можуть одночасно виконувати ділення з плаваючою точкою, так як цим займається лише одна частина процесора. Але, якщо одному з апаратних потоків потрібно ділити, а інший зайнятий перемножуванням чисел, їм, як правило, вдається робити це паралельно, оскільки дані операції виконують різні частини ядра.

3. Конструкції паралельного програмування

- взаимоисключение, умовна синхронізація, примітиви синхронізації;
- критичні секції, види блокувань.

Паралельна програма містить кілька процесів, ра-ботающих спільно над виконанням деякої задами. Кожен процес - це послідовна програма, а томнее - послідовність операторів, що виконуються один за іншим. Послідовна програма має один потік управління, а паралельна - кілька.

Спільна робота процесів паралельної програми здійс-нюється за допомогою їх воші моді істин я. Взаємодія програмується з застосуванням поділюваних змінних або пересилання повідомлень. Якщо використовуються колективні змінні. один процес здійснює запис у змінну, считываемую іншим процесом. При пересиланні повідомлень один процес відправляє повідомлення, яке отримує інший. При будь-якому вигляді взаємодії процесів необхідна взаємна синхронізація. Існують два основних види синхронізації взаємне виключення і умовна синхронізація.



Взаємне виключення забезпечує, щоб критичні секції операторів не виконувалися одночасно. Умовна синхронізація затримує процес до тих пір, поки не виконається певну умову.

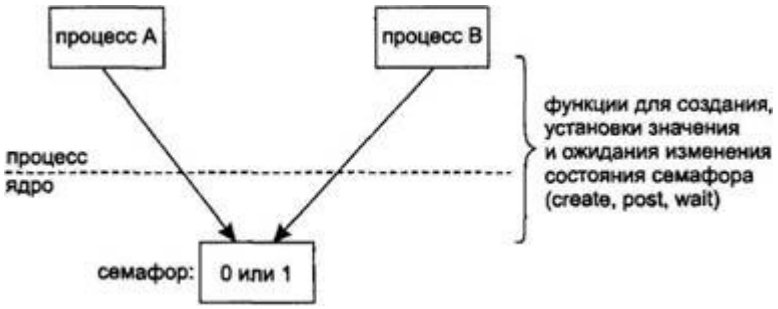
Апаратні методи організації взаємовиключення.

В чому основна проблема програмних методів взаємовиключення? Неможливо гарантувати нерозривність виконання окремих дій: програма може перерватися в будь-який момент і можливі довільні послідовності операцій у паралельній програмі (недетерминизм). Одне з рішень – введення в апаратуру (або в системний шар) спеціальних конструкцій, що володіють властивістю нерозривності (автомарности).

Примітиви синхронізації.

Семафори— об'єкт, що обмежує кількість потоків, які можуть увійти в заданий ділянку коду. Визначення введено ЭдсгеромДейкстрой. Семафори використовуються при передачі даних через поділювану пам'ять.

М'ютекси — це найпростіші двійкові семафори, які можуть знаходитися в одному з двох станів — або зазначеному неотмеченном (відкритий і закритий відповідно). Коли який-небудь потік, що належить будь-якого процесу, стає власником об'єкта mutex, останній переводиться в неотмеченное стан.



Критична секція(CriticalSection) це ділянка коду, в якому потік (thread) отримує доступ до ресурсу (наприклад змінна), який доступний з інших потоків. Об'єкт критична секція забезпечує синхронізацію.

Спинлок (англ. Spinlock — циклічна блокування) — низькорівневий примітив синхронізації, застосовуваний в багатопроцесорних системах для реалізації взаємного виключення.

Монітори — у мовах програмування, високорівневий механізм взаємодії і синхронізації процесів, що забезпечує доступ до неразделяемым ресурсів.[1] Підхід до синхронізації двох або більше комп'ютерів, що використовують загальний ресурс, зазвичай апаратуру або набір змінних.

Умовна змінна — примітив синхронізації, що забезпечує блокування одного або декількох потоків до моменту надходження сигналу від іншого потоку про виконання деякої умови або до закінчення максимального проміжку часу очікування. Умовні змінні використовуються разом з асоційованим мьютексом і є елементом деяких видів моніторів.

Readers-Writeslock (RWLock)– необхідний для асиметричної схеми доступу до ресурсу.

Блокування - це механізм синхронізації дозволяє забезпечити винятковий доступ до поділюваному ресурсу між кількома потоками. Блокування один із способів забезпечити політику управління розпаралелюванням.

Види блокувань.

В основному, використовується м'яка блокування, при цьому передбачається, що кожен потік намагається отримати блокування перед доступом до відповідного поділюваному ресурсу. В деяких системах надається механізм обов'язкового блокування,

при його використанні спроба несанкціонованого доступу до заблокованого ресурсу буде перервана, через створення виключення в потоці, який намагався отримати доступ.

Семафор найпростіший тип блокування. З точки зору доступу до даних не робиться ніяких відмінностей між режимами доступу: загальним (тільки читання) або ексклюзивним (читання і запис). В режимі загального доступу кілька потоків можуть запросити для блокування доступу до даних у режимі тільки читання. Також використовується ексклюзивний режим доступу в алгоритмах оновлення та видалення.

Типи блокувань розрізняють стратегії блокування продовження виконання потоку. У більшості реалізацій запит блокування перешкоджає подальшому виконанню потоку поки не з'явиться доступ до заблокованої ресурсу.

Спинлок це блокування яка очікує у циклі поки не з'явиться доступ. Така блокування дуже ефективна якщо потік очікує блокування незначний інтервал часу, це дозволяє уникнути надмірної перепланування потоків. Витрати на очікування доступу будуть значними при тривалому утриманні блокування одним з потоків.

Для ефективної реалізації механізму блокування потрібна підтримка на апаратному рівні. Апаратна підтримка може бути реалізована у вигляді однієї або декількох атомарних операціях таких як "test-and-set", "fetch-and-add" або "compare-and-swap". Такі інструкції дозволяють без переривань перевірити, що блокування вільна і якщо це так то зайняти блокування.

В однопроцесорних системах є можливість виконувати інструкції без апаратних переривань використовуючи спеціальні інструкції або префікси інструкції, які тимчасово відключають переривання, але такий підхід не працює в багатопроцесорних системах зі спільною пам'яттю. Повна підтримка блокувань в багатопроцесорному оточенні може вимагати досить складною апаратної і програмної підтримки, зі значними проблемами синхронізації.