

ЛЕКЦІЯ 3. ПЛАНУВАННЯ ПРОЦЕСІВ І ПОТОКІВ

- 3.1. Базові поняття процесів і потоків
- 3.2. Багатопотоковість та її реалізація
- 3.3. Стани процесів і потоків
- 3.4. Загальні принципи планування
- 3.5. Види планування
- 3.6. Алгоритми планування
- 3.7. Реалізація планування у Windows

3.1. Базові поняття процесів і потоків

У сучасній операційній системі одночасно виконуються код ядра (що належить до його різних підсистем) і код програм користувача. При цьому відбуваються різні дії: одні програми і підсистеми виконують інструкції процесора, інші зайняті введенням-виведенням, ще деякі очікують на запити від користувача або інших застосувань. Для спрощення керування цими діями в системі доцільно виділити набір елементарних активних елементів і визначити інтерфейс взаємодії ОС із цими елементами. **Коли активний елемент системи зв'язати із програмою, що виконується, ми прийдемо до поняття процесу.** Дамо попереднє означення процесу.

Під процесом розуміють абстракцію ОС, яка об'єднує все необхідне для виконання однієї програми в певний момент часу.

Програма – це деяка послідовність машинних команд, що зберігається на диску, в разі необхідності завантажується у пам'ять і виконується. Можна сказати, що **під час виконання програму представляє процес.**

Однозначна відповідність між програмою і процесом встановлюється тільки в конкретний момент часу: один процес у різний час може виконувати код декількох програм, код однієї програми можуть виконувати декілька процесів одночасно.

Для успішного виконання програми потрібні певні ресурси. До них належать:

- ресурси, необхідні для послідовного виконання програмного коду (передусім процесорний час);
- ресурси, що дають можливість зберігати інформацію, яка забезпечує виконання програмного коду (реєстри процесора, оперативна пам'ять тощо).

Ці групи ресурсів визначають дві складові частини процесу:

- послідовність виконуваних команд процесора;
- набір адрес пам'яті (адресний простір), у якому розташовані ці команди і дані для них.

Виділення цих частин виправдане ще й тим, що **в рамках одного адресного простору може бути кілька паралельно виконуваних послідовностей команд, що спільно використовують одні й ті ж самі дані. Необхідність розмежування послідовності команд і адресного простору підводить до поняття потоку.**

Потоком (потік керування, нитка, thread) називають набір послідовно виконуваних команд процесора, які використовують загальний адресний простір процесу.

Оскільки в системі може одночасно бути багато потоків, завданням ОС є організація перемикання процесора між ними і планування їхнього виконання. У багатопроцесорних системах код окремих потоків може виконуватися на окремих процесорах.

Тепер можна дати ще одне означення процесу.

Процесом називають сукупність одного або декількох потоків і захищеного адресного простору, у якому ці потоки виконуються.

Захищеність адресного простору процесу є його найважливішою характеристикою.

Код і дані процесу не можуть бути прямо прочитані або перезаписані нішим процесом; у такий спосіб захищаються від багатьох програмних помилок і спроб несанкціонованого доступу. Природно, що неприпустимим є тільки *прямий* доступ (наприклад, запис у пам'ять за допомогою простої інструкції перенесення даних); **обмін даними між процесами принципово можливий, але для цього мають бути використані спеціальні засоби, які називають засобами міжпроцесової взаємодії**. Такі засоби складніші за прямий доступ і працюють повільніше, але при цьому забезпечують захист від випадкових помилок у разі доступу до даних.

На відміну від процесів *потоків розпоряджаються загальною пам'яттю*. **Дані потоку не захищені від доступу до них інших потоків за умови, що всі вони виконуються в адресному просторі одного процесу**. Це надає додаткові можливості для розробки застосувань, але ускладнює програмування.

Захищений адресний простір процесу задає абстракцію виконання коду на окремій машині, а потік забезпечує абстракцію послідовного виконання команд на одному виділеному процесорі.

Адресний простір процесу не завжди відповідає адресам оперативної пам'яті. Наприклад, у нього можуть відображатися файли або реєстри контролерів введення-виведення, тому запис за певною адресою в цьому просторі призведе до запису у файл або до виконання операції введення-виведення. Таку технологію називають *відображенням у пам'ять* (memory mapping).

Максимально можлива кількість процесів (захищених адресних просторів) і потоків, які в них виконуються, може варіюватися в різних системах.

У більшості сучасних ОС (таких, як лінія Windows, сучасні версії UNIX) може бути багато процесів, а в адресному просторі кожного процесу – багато потоків. Ці системи підтримують багатопотоковість або реалізують модель потоків. Процес у такій системі називають багатопотоковим процесом.

Надалі для позначення *послідовності виконуваних команд вживатимемо термін «потік»*, за винятком ситуацій, коли обговорюватиметься реалізація моделі процесів.

До елементів процесу належать:

- захищений адресний простір;
- дані, спільні для всього процесу (ці дані можуть спільно використовувати всі його потоки);
- інформація про використання ресурсів (відкриті файли, мережні з'єднання тощо);
- інформація про потоки процесу.

Потік містить такі елементи:

- стан процесора (набір поточних даних із його реєстрів), зокрема лічильник поточної інструкції процесора;
- стек потоку (ділянка пам'яті, де перебувають локальні змінні потоку й адреси повернення функцій, що викликані у його коді).

3.2. Багатопотоковість та її реалізація

Використання декількох потоків у застосуванні означає внесення в нього паралелізму (concurrency). **Паралелізм – це одночасне (з погляду прикладного програміста) виконання дій різними фрагментами коду застосування**. Така одночасність може бути реалізована на одному процесорі шляхом перемикавання задач (випадок *псевдопаралелізму*), а може ґрунтуватися на паралельному виконанні коду на декількох процесорах (випадок *справжнього паралелізму*). Потоки абстрагують цю відмінність, даючи

можливість розробляти застосування, які в однопроцесорних системах використовують псевдопаралелізм, а при доданні процесорів – справжній паралелізм (такі застосування масштабуються зі збільшенням кількості процесорів).

Можна виділити такі основні види паралелізму:

- паралелізм багатопроцесорних систем;
- паралелізм операцій введення-виведення;
- паралелізм взаємодії з користувачем;
- паралелізм розподілених систем.

Перший з них є справжнім паралелізмом, тому що у багатопроцесорних системах інструкції виконують декілька процесорів одночасно. Інші види паралелізму можуть виникати і в однопроцесорних системах тоді, коли для продовження виконання програмного коду необхідні певні зовнішні дії.

Паралелізм операцій введення-виведення

Під час виконання операції введення-виведення (наприклад, обміну даними із диском) низькорівневий код доступу до диска і код застосування не можуть виконуватись одночасно. У цьому разі застосуванню треба почекати завершення операції введення-виведення, звільнивши на цей час процесор. Природним вважається зайняти на цей час процесор інструкціями іншої задачі.

Багатопотокове застосування може реалізувати цей вид паралелізму через створення нових потоків, які виконуватимуться, коли поточний потік очікує операції введення-виведення. Так реалізується асинхронне введення-виведення, коли застосування продовжує своє виконання, не чекаючи на завершення операцій введення-виведення.

Паралелізм взаємодії з користувачем

Під час інтерактивного сеансу роботи користувач може виконувати різні дії із застосуванням (і очікувати негайної реакції на них) до завершення обробки попередніх дій. Наприклад, після запуску команди «друкування документа» він може негайно продовжити введення тексту, не чекаючи завершення друкування.

Щоб розв'язати це завдання, можна виділити окремі потоки для безпосередньої взаємодії із користувачем (наприклад, один потік може очікувати введення з клавіатури, інший – від миші, додаткові потоки – відображати інтерфейс користувача). Основні задачі застосування (розрахунки, взаємодія з базою даних тощо) у цей час виконуватимуть інші потоки.

Паралелізм розподілених застосувань

Розглянемо серверне застосування, яке очікує запити від клієнтів і виконує дії у відповідь на запит. Якщо клієнтів багато, запити можуть надходити часто, майже водночас. Якщо тривалість обробки запиту перевищує інтервал між запитами, сервер буде змушений поміщати запити в чергу, внаслідок чого знижується продуктивність. При цьому використання потоків дає можливість організувати паралельне обслуговування запитів, коли основний потік приймає запити, відразу передає їх для виконання іншим потокам і очікує нових.

Назвемо проблеми, які можуть бути вирішені за допомогою потоків.

- Використання потоків дає змогу реалізувати різні види паралелізму і дозволяє застосуванню масштабуватися із ростом кількості процесорів.
- Для підтримки потоків потрібно менше ресурсів, ніж для підтримки процесів (наприклад, немає необхідності виділяти для потоків адресний простір).

- Для обміну даними між потоками може бути використана спільна пам'ять (адресний простір їхнього процесу). Це ефективніше, ніж застосовувати засоби міжпроцесової взаємодії.

Незважаючи на перелічені переваги, використання потоків не є універсальним засобом розв'язання проблем паралелізму, і пов'язане з деякими труднощами.

- Розробляти і налагоджувати багатопотокові програми складніше, ніж звичайні послідовні програми; досить часто впровадження багатопотоковості призводить до зниження надійності застосувань. Організація спільного використання адресного простору декількома потоками вимагає, щоб програміст мав високу кваліфікацію.
- Використання потоків може спричинити зниження продуктивності застосувань. Переважно це трапляється в однопроцесорних системах (наприклад, у таких системах спроба виконати складний розрахунок паралельно декількома потоками призводить лише до зайвих витрат на перемикання між потоками, кількість виконаних корисних інструкцій залишиться тією ж самою).

Переваги і недоліки використання потоків потрібно враховувати під час виконання будь-якого програмного проекту. Насамперед доцільно розглядати можливість розв'язати задачу в рамках моделі процесів. При цьому, однак, варто брати до уваги не лише поточні вимоги замовника, але й необхідність розвитку застосування, його масштабування тощо. Можливо, з урахуванням цих факторів використання потоків буде виправдане.

Перш ніж розглянути основні підходи до реалізації моделі потоків, дамо означення важливих понять потоку користувача і потоку ядра.

Потік користувача – це послідовність виконання команд в адресному просторі процесу. Ядро ОС не має інформації про такі потоки, вся робота з ними виконується в режимі користувача. Засоби підтримки потоків користувача надають спеціальні системні бібліотеки; вони доступні для прикладних програмістів у вигляді бібліотечних функцій. Бібліотеки підтримки потоків у наш час звичайно реалізують набір функцій, визначений стандартом POSIX (відповідний розділ стандарту називають POSIX.lb); тут прийнято говорити про підтримку *потоків POSIX*.

Потік ядра – це послідовність виконання команд в адресному просторі ядра. Потоками ядра управляє ОС, перемикання ними можливе тільки у привілейованому режимі. Є потоки ядра, які відповідають потокам користувача, і потоки, що не мають такої відповідності.

Співвідношення між двома видами потоків визначає реалізацію моделі потоків. Є кілька варіантів такої реалізації (схем багатопотоковості); розглянемо найважливіші з них.

Схема багатопотоковості М:1 (є найранішою) реалізує багатопотоковість винятково в режимі користувача. При цьому кожен процес може містити багато потоків користувача, однак про наявність цих потоків ОС не відомо, вона працює тільки із процесами. За планування потоків і перемикання контексту відповідає бібліотека підтримки потоків. Схема вирізняється ефективністю керування потоками (для цього немає потреби переходити в режим ядра) і не потребує для реалізації зміни ядра ОС. Проте нині її практично не використовують через два **суттєвих недоліки**, що не відповідають ідеології багатопотоковості.

- Схема М:1 не дає змоги скористатися багатопроцесорними архітектурами, оскільки визначити, який саме код виконуватиметься на кожному із процесорів, може тільки ядро ОС. У результаті всі потоки одного процесу завжди виконуватимуться на одному процесорі.
- Оскільки системні виклики обробляються на рівні ядра ОС, блокувальний системний виклик (наприклад, виклик, який очікує введення даних користувачем)

зупинятиме всі потоки процесу, а не лише той, що зробив цей виклик.

Схема багатопотоковості 1:1 ставить у відповідність кожному потоку користувача один потік ядра. У цьому разі планування і перемикання контексту зачіпають лише потоки ядра, у режимі користувача ці функції не реалізовані. Оскільки ядро ОС має інформацію про потоки, ця схема вільна від недоліків попередньої (різні потоки можуть виконуватися на різних процесорах, а при зупиненні одного потоку інші продовжують роботу). Вона проста і надійна в реалізації і **сьогодні є найпоширенішою**. Хоча схема передбачає, що під час керування потоками треба постійно перемикатися між режимами процесора, на практиці втрата продуктивності внаслідок цього виявляється незначною.

Схема багатопотоковості M:N. У цій схемі присутні як потоки ядра, так і потоки користувача, які відображаються на потоки ядра так, що один потік ядра може відповідати декільком потокам користувача. Число потоків ядра може бути змінено програмістом для досягнення максимальної продуктивності. Розподіл потоків користувача по потоках ядра виконується в режимі користувача, планування потоків ядра – у режимі ядра. *Схема є складною в реалізації і сьогодні здає свої позиції схемі 1:1.*

3.3. Стани процесів і потоків

Для потоку дозволені такі стани:

- створення (new) – потік перебуває у процесі створення;
- виконання (running) – інструкції потоку виконує процесор (у конкретний момент часу на одному процесорі тільки один потік може бути в такому стані);
- очікування (waiting) – потік очікує деякої події (наприклад, завершення операції введення-виведення); такий стан називають також заблокованим, а потік – припиненим;
- готовність (ready) – потік очікує, що планувальник перемкне процесор на нього, при цьому він має всі необхідні йому ресурси, крім процесорного часу;
- завершення (terminated) – потік завершив виконання (якщо при цьому його ресурси не були вилучені з системи, він переходить у додатковий стан – стан зомбі).

Можливі переходи між станами потоку зображено на рис. 3.1.

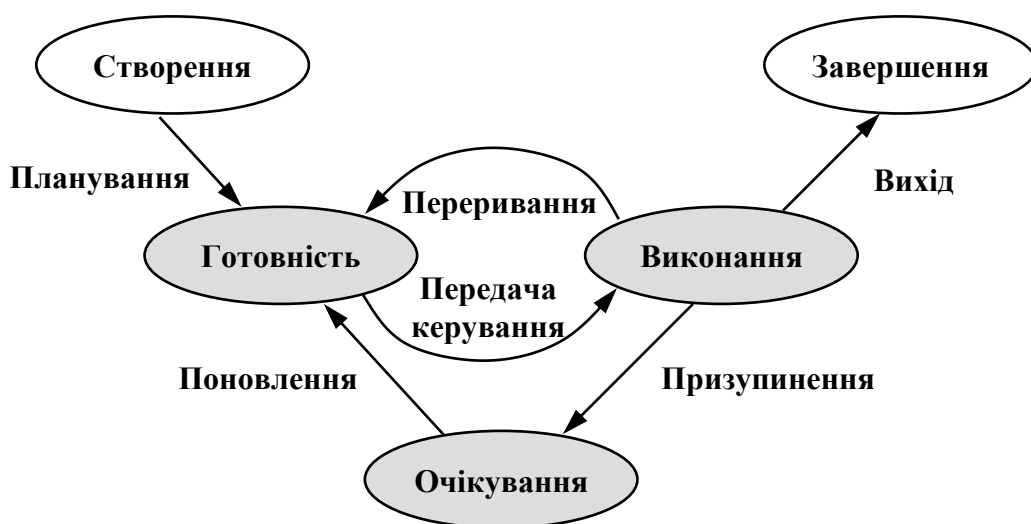


Рис. 3.1. Стани потоку

Перехід потоків між станами очікування і готовності реалізовано на основі *планування задач, або планування потоків*. Під час планування потоків визначають, який з потоків треба відновити після завершення операції введення-виведення, як організувати очікування подій у системі.

Для здійснення переходу потоків між станами готовності та виконання необхідне *планування процесорного часу*. На основі алгоритмів такого планування визначають, який з готових потоків потрібно виконувати в конкретний момент, коли потрібно перервати виконання потоку, щоб перемкнутися на інший готовий потік тощо. Відносно систем, які реалізують модель процесів, прийнято говорити про стани процесів, а не потоків, і про планування процесів; фактично стани процесу в цьому разі однозначно відповідають станам його єдиного потоку.

У багатопотокових системах також можна виділяти стани процесів. Наприклад, у багатопотоковості, реалізованій за схемою M:1, потоки змінюють свої стани в режимі користувача, а процеси – у режимі ядра.

Одним із основних завдань операційної системи є розподіл ресурсів між процесами і потоками. Для керування розподілом ресурсів ОС повинна підтримувати структури даних, які містять інформацію, що описує процеси, потоки і ресурси. До таких структур даних належать:

- таблиці розподілу ресурсів: таблиці пам'яті, таблиці введення-виведення, таблиці файлів тощо;
- таблиці процесів і таблиці потоків, де міститься інформація про процеси і потоки, присутні у системі в конкретний момент.

Керуючий блок потоку (Thread Control Block, TCB) відповідає активному потоку, тобто тому, який перебуває у стані готовності, очікування або виконання. Цей блок може містити таку інформацію:

- ідентифікаційні дані потоку (зазвичай його унікальний ідентифікатор);
- стан процесора потоку: користувацькі регістри процесора, лічильник інструкцій, покажчик на стек;
- інформацію для планування потоків.

Таблиця потоків – це зв'язний список або масив керуючих блоків потоку. Вона розташована в захищеній області пам'яті ОС.

Керуючий блок процесу (Process Control Block, PCB) відповідає процесу, що присутній у системі. Такий блок може містити:

- ідентифікаційні дані процесу (унікальний ідентифікатор, інформацію про інші процеси, пов'язані з даним);
- інформацію про потоки, які виконуються в адресному просторі процесу (наприклад, покажчики на їхні керуючі блоки);
- інформацію, на основі якої можна визначити права процесу на використання різних ресурсів (наприклад, ідентифікатор користувача, який створив процес);
- інформацію з розподілу адресного простору процесу;
- інформацію про ресурси введення-виведення та файли, які використовує процес.

Зазначимо, що для систем, у яких реалізована модель процесів, у керуючому блоці процесу зберігають не посилання на керуючі блоки його потоків, а інформацію, необхідну безпосередньо для його виконання (лічильник інструкцій, дані для планування тощо).

Таблицю процесів організовують аналогічно до таблиці потоків. Як елементи в ній зберігатимуться керуючі блоки процесів.

Сукупність інформації, що відображає процес у пам'яті, називають *образом процесу* (process image), а всю інформацію про потік – *образом потоку* (thread image). До образу процесу належать:

- керуючий блок процесу;

- програмний код користувача;
- дані користувача (глобальні дані програми, загальні для всіх потоків);
- інформація образів потоків процесу.

Програмний код користувача, дані користувача та інформація про потоки завантажуються в адресний простір процесу. Образ процесу звичайно не є безперервною ділянкою пам'яті, його частини можуть вивантажуватися на диск.

До образу потоку належать:

- керуючий блок потоку;
- стек ядра (стек потоку, який використовується під час виконання коду потоку в режимі ядра);
- стек користувача (стек потоку, доступний у користувацькому режимі).

3.4. Загальні принципи планування

З погляду планування виконання потоку можна зобразити як цикл чергування періодів обчислень (використання процесора) і періодів очікування введення-виведення. Інтервал часу, упродовж якого потік виконує тільки інструкції процесора, називають інтервалом використання процесора (CPU burst), інтервал часу, коли потік очікує введення-виведення, – інтервалом введення-виведення (I/O burst). Найчастіше ці інтервали мають довжину від 2 до 8 мс.


Потоки, які більше часу витрачають на обчислення і менше – на введення-виведення, називають **обмеженими можливостями процесора (CPU bound)**. Вони активно використовують процесор. Основною їхньою характеристикою є час, витрачений на обчислення, інтервали використання процесора для них довші. Потоки, які більшу частину часу перебувають в очікуванні введення-виведення, називають **обмеженими можливостями введення-виведення (I/O bound)**. Такі потоки завантажують процесор значно менше, а середня довжина інтервалу використання процесора для них невелика. Що вища тактова частота процесора, то більше потоків можна віднести до другої категорії.

Потік, обмежений процесором (перемножування матриць)



Потік, обмежений введенням-виведенням (текстовий редактор)



 Інтервал введення-виведення (I/O burst)


 Інтервал використання процесора (CPU burst)

Рис. 3.2. Класифікація потоків з погляду планування

Слід розрізняти **механізми** і **політику** планування. До механізмів планування належать засоби перемикання контексту, засоби синхронізації потоків тощо, до політики планування – засоби визначення моменту часу, коли необхідно перемкнути контекст. Ту частину системи, яка відповідає за політику планування, називають **планувальником (scheduler)**, а алгоритм, що використовують при цьому, – **алгоритмом планування (scheduling algorithm)**.

Є різні критерії оцінки політики планування, одні з них застосовні для всіх систем, інші – лише для пакетних систем або лише для інтерактивних.

Сьогодні найчастіше використовують три критерії оцінки досягнення мети.

- **Мінімальний час відгуку.** Це найважливіший критерій для інтерактивних систем. Під часом відгуку розуміють час між запуском потоку (або введенням користувачем інтерактивної команди) і отриманням першої відповіді. Для сучасних систем прийнятним часом відгуку вважають 50-150 мс.
- **Максимальна пропускна здатність.** Це кількість задач, які система може виконувати за одиницю часу (наприклад, за секунду). Такий критерій доцільно застосовувати у пакетних системах; в інтерактивних системах він може бути використаний для фонових задач. Щоб підвищити пропускну здатність, необхідно:
 - скорочувати час даремного навантаження (наприклад, час, необхідний для перемикавання контексту);
 - ефективніше використати ресурси (для того, щоб ані процесор, ані пристрої введення-виведення не простоювали).
- Третім критерієм є справедливість, яка полягає в тому, що процесорний час потокам виділяють відповідно до їхньої важливості. Справедливість забезпечує такий розподіл процесорного часу, що всі потоки просуваються у своєму виконанні, і жоден не простоє.

Відзначимо, що реалізація справедливої політики планування не завжди призводить до зменшення середнього часу відгуку. Іноді для цього потрібно зробити систему менш справедливою.

3.5. Види планування

Розрізняють планування *довготермінове* (long-term scheduling), *середньотермінове* (medium-term scheduling) і *короткотермінове* (short-term scheduling).

Довготермінове планування

Засоби довготермінового планування визначають, яку з програм треба завантажити у пам'ять для виконання. Таке планування називають також статичним, оскільки воно не залежить від поточного стану системи. Така стратегія ґрунтується і на психології користувачів, які, почуваючи себе некомфортно в перевантаженій системі, можуть переривати роботу з нею, що призводить до зниження навантаження.

Середньотермінове планування

Засоби середньотермінового планування керують переходом потоків із призупиненого стану в стан готовності й назад. Відразу ж зазначимо, що керуючі блоки готових до виконання потоків організуються у пам'яті в структуру, яку називають чергою готових потоків (ready queue).

Перехід потоку в призупинений стан можуть викликати такі фактори:

- очікування операції введення-виведення;
- очікування закінчення виконання іншого потоку (приєднання);
- блокування потоку через необхідність його синхронізації з іншими потоками.

Короткотермінове планування

Короткотермінове планування, або *планування процесора* (CPU scheduling), є найважливішим видом планування. Воно дає змогу відповісти на два базових запитання.

- Коли перервати виконання потоку?
- Якому потокові з числа готових до виконання потрібно передати процесор у цей момент?

Короткотерміновий планувальник – це підсистема ОС, яка в разі необхідності

перериває активний потік і вибирає з черги готових потоків той, що має виконуватися. До його продуктивності ставлять найвищі вимоги, бо він отримує керування дуже часто. Виділяють також *диспетчер* (dispatcher), який безпосередньо передає керування вибраному потокові (перемикає контекст).

Формат черги готових потоків залежить від реалізації короткотермінового планування. Така черга може бути організована за принципом FIFO, бути чергою із пріоритетами, деревом або невпорядкованим зв'язним списком.

Усі стратегії й алгоритми планування, які ми будемо розглядати далі, належать до короткотермінового планування.

3.6. Алгоритми планування

Можна сказати, що алгоритми планування реалізують політику планування. Залежно від стратегії планування, яку реалізують *алгоритми*, їх поділяють на *витісняльні та невитісняльні*. Витісняльні алгоритми переривають потоки під час їхнього виконання, невитісняльні – не переривають. Деякі алгоритми відповідають лише одній із цих стратегій, інші можуть мати як витісняльний, так і невитісняльний варіанти реалізації.

Планування за принципом FIFO

Найпростіший («наївний») невитісняльний алгоритм, у якому потоки ставлять на виконання в порядку їхньої появи у системі й виконують до переходу в стан очікування, явної передачі керування або завершення. Чергу готових потоків при цьому організовують за принципом FIFO, тому алгоритм називають алгоритмом FIFO.

Як тільки в системі створюється новий потік, його керуючий блок додається у хвіст черги. Коли процесор звільняється, його надають потоку з голови черги.

Кругове планування

Найпростішим для розуміння і найсправедливішим витісняльним алгоритмом є *алгоритм кругового планування* (round-robin scheduling).

Кожному потокові виділяють інтервал часу, який називають *квантом часу* (time quantum, time slice) і упродовж якого цьому потокові дозволено виконуватися. Коли потік усе ще виконується після вичерпання кванта часу, його переривають і перемикають процесор на виконання інструкцій іншого потоку. Коли він блокується або закінчує своє виконання до вичерпання кванта часу, процесор теж передають іншому потокові. Довжина кванта часу для всієї системи однакова.

Єдиною характеристикою, яка впливає на роботу алгоритму, є довжина кванта часу. Задавання надто короткого кванта часу призводить до того, що відбувається дуже багато перемикань контексту, і значний відсоток процесорного часу витрачається не на корисну роботу, а на ці перемикання. З іншого боку, задавання надто довгого кванта хоча й заощаджує процесорний час, але спричиняє до зниження часу відгуку на інтерактивні запити, бо якщо десять користувачів одночасно натиснуть клавішу, то десять потоків потраплять у список готових, внаслідок чого останній з них очікуватиме десять довгих квантів часу. У випадку з квантом нескінченної довжини кругове планування зводиться до алгоритму FIFO (усі потоки встигають заблокуватися або закінчитися до вичерпання кванта). На практиці рекомендують встановлювати довжину кванта в 10-100 мс.

Планування із пріоритетами

Основна ідея проста: кожному потокові надають пріоритет, при цьому на виконання

ставитиметься потік із найвищим пріоритетом із черги готових потоків. Пріоритети можуть надаватися потокам статично або динамічно.

Одним із підходів до реалізації планування із пріоритетами є алгоритм *багаторівневих черг* (multilevel queues). У цьому разі організовують кілька черг для груп потоків із різними пріоритетами (потоки кожної групи звичайно мають різне призначення, можуть бути групи фонових потоків, інтерактивних тощо).

Для різних черг можна використовувати різні алгоритми планування, крім того, кожній черзі може бути виділена певна частка процесорного часу.

Розподіл пріоритетів є складним завданням, невдале його розв'язання може призвести до того, що потоки процесів із низьким пріоритетом чекатимуть дуже довго. Наприклад, у 1973 році в Массачусетському технологічному інституті була зупинена машина, на якій знайшли процес із низьким пріоритетом – він був поставлений у чергу на виконання в 1967 році і з того часу так і не зміг запуститися. Таку ситуацію називають *голодуванням* (starvation).

Планування на підставі характеристик подальшого виконання

Важливим класом алгоритмів планування з пріоритетами є алгоритми, в яких рішення про вибір потоку для виконання приймають на підставі знання або оцінки характеристик подальшого його виконання.

Насамперед слід відзначити алгоритм *«перший – із найкоротшим часом виконання»* (Shortest Time to Completion First, STCF), коли з кожним потоком пов'язують тривалість наступного інтервалу використання ним процесора і для виконання щоразу вибирають той потік, у якого цей інтервал найкоротший. У результаті потоки, що захоплюють процесор на короткий час, отримують під час планування перевагу і швидше виходять із системи.

Алгоритм STCF є теоретично оптимальним за критерієм середнього часу відгуку, тобто можна довести, що для вибраної групи потоків середній час відгуку в разі застосування цього алгоритму буде мінімальним порівняно з будь-яким іншим алгоритмом. На жаль, для короткотермінового планування реалізувати його неможливо, тому що ця реалізація потребує передбачення очікуваних характеристик. Для довготермінового планування його використовують досить часто (у цьому разі, ставлячи задачу на виконання, оператор повинен вказати очікуваний момент її завершення, на який система буде зважати під час вибору). Зазначимо також, що оптимальність такого алгоритму невіддільна від його «несправедливості» до потоків із довгими інтервалами використання процесора.

Витісняльним аналогом STCF є алгоритм *«перший – із найкоротшим часом виконання, що залишився»* (Shortest Remaining Time to Completion First, SRTCF). Його відмінність від STCF полягає в тому, що, коли в чергу готових потоків додають новий, у якого наступний інтервал використання процесора коротший, ніж час, що залишився до завершення виконання поточного потоку, поточний потік переривається, і на його місце стає новий потік.

Багаторівневі черги зі зворотним зв'язком

Алгоритм *багаторівневих черг зі зворотним зв'язком* (multilevel feedback queues) є найуніверсальнішим алгоритмом планування (за допомогою налаштування параметрів його можна звести майже до будь-якого іншого алгоритму), але при цьому одним із найскладніших у реалізації.

З погляду організації структур даних цей алгоритм схожий на звичайний алгоритм

багаторівневих черг: є кілька черг готових потоків із різним пріоритетом, при цьому потоки черги із нижчим пріоритетом виконуються, тільки коли всі черги верхнього рівня порожні.

Відмінності між двома алгоритмами полягають у тому, що:

- потокам дозволено переходити з рівня на рівень (із черги в чергу);
- потоки в одній черзі об'єднуються не за пріоритетом, а за довжиною інтервалу використання процесора, потоки із коротшим інтервалом перебувають у черзі з більшим пріоритетом.

Усередині всіх черг, крім найнижчої, використовують кругове планування (у найнижчій працює FIFO-алгоритм). Різні черги відповідають різній довжині кванта часу – що вищий пріоритет, то коротший квант (звичайно довжина кванта для сусідніх черг зменшується удвічі). Якщо потік вичерпав свій квант часу, він переміщається у хвіст черги із нижчим пріоритетом (і з довшим квантом). У результаті потоки з коротшими інтервалами (наприклад, обмежені введенням-виведенням) залишаються з високим пріоритетом, а потоки з довшими інтервалами подовжують свій квант часу. Можна також автоматично переміщати потоки, які давно не отримували керування, із черги нижнього рівня на рівень вище.

Лотерейне планування

Ідея лотерейного планування полягає у тому, що:

- потік отримує деяку кількість лотерейних квитків, кожен з яких дає право користуватися процесором упродовж часу T ;
- планувальник через проміжок часу T вибирає випадково один лотерейний квиток;
- потік, «що виграв», дістає керування до наступного розіграшу.

Лотерейне планування дає змогу:

- емулювати кругове планування, видавши кожному потокові однакову кількість квитків;
- емулювати планування із пріоритетами, розподіляючи квитки відповідно до пріоритетів потоків;
- емулювати SRTCF – давати коротким потокам більше квитків, ніж довгим (якщо потік отримав хоча б один квиток, він не голодуватиме);
- забезпечити розподіл процесорного часу між потоками – дати кожному потокові кількість квитків, пропорційну до частки процесорного часу, який потрібно йому виділити;
- динамічно міняти пріоритети, відбираючи і додаючи квитки по ходу.

Насправді лотерейне планування використовує той факт, що вся ідеологія планування значною мірою є евристичною, оскільки не можна точно передбачити характер поведінки потоків у системі.

3.7. Реалізація планування у Windows

Планування потоків у ядрі

Ядро Windows розв'язує під час планування дві основні задачі:

- облік відносних пріоритетів, присвоєних кожному потокові;
- мінімізацію часу відгуку інтерактивних застосувань.

Базовою одиницею планування є потік. Під час планування ядро не розрізняє потоки різних процесів, воно має справу з пріоритетами потоків, готових до виконання в певний момент часу.

Під час планування ядро працює з мінімальними версіями потоків (блоками KTHREAD). У них зберігається така інформація, як загальний час виконання потоку, його базовий і поточний пріоритет, диспетчерський стан потоку (готовність, очікування, виконання тощо).

Пріоритети потоків і процесів

Для визначення порядку виконання потоків диспетчер ядра використовує систему пріоритетів. Кожному потокові присвоюють пріоритет, заданий числом у діапазоні від 1 до 31 (що більше число, то вище пріоритет). Пріоритети реального часу – 16-31; їх резервує система для дій, час виконання яких є критичним чинником. Динамічні пріоритети – 1-15; вони можуть бути присвоєні потокам застосувань користувача.

Ядро системи може надати потоку будь-який динамічний пріоритет. Win32 API не дає можливості зробити це з цілковитою точністю, у ньому використовують дворівневу систему, яка зачіпає як процес, так і його потоки: спочатку процесу присвоюють *клас пріоритету*, а потім потокам цього процесу – відносний пріоритет, який відраховують від класу пріоритету процесу (називаного ще базовим пріоритетом). Під час виконання відносний пріоритет може змінюватися.

Розрізняють такі класи пріоритету процесів: реального часу (real-time, приблизно відповідає пріоритету потоку 24); високий (high, 13); нормальний (normal, 8); невикористовуваний (idle, 4).

Відносні пріоритети потоку бувають такі: найвищий (+2 до базового); вище за нормальний (+1 до базового); нормальний (дорівнює базовому); нижче за нормальний (-1 від базового); найнижчий (-2 від базового).

Є два додаткових модифікатори відносного пріоритету: критичний за часом (time-critical) і невикористовуваний (idle). Перший модифікатор тимчасово задає для потоку пріоритет 15 (найвищий динамічний пріоритет), другий аналогічним чином задає пріоритет 1.

Особливості задавання кванта часу

Важливою характеристикою системи є довжина кванта часу. Розрізняють короткі й довгі кванти, для яких можна задати змінну та фіксовану довжину.

У Windows інтерактивно можна задавати таку довжину кванта (вибирають Settings (Параметри) у групі Performance (Быстродействие) на вкладці Advanced (Дополнительно) вікна властивостей My Computer (Свойства системы)):

- короткі кванти змінної довжини (вкладка Advanced (Дополнительно), перемикач Programs (Програми) у групі властивостей Processor Scheduling (Распределение времени процессора)). Можлива довжина кванта – 10 або 30 мс, при цьому застосування, з яким починає працювати користувач, автоматично переходить до використання довгих квантів. Ця установка надає перевагу інтерактивним процесам;
- довгі кванти фіксованої довжини (вкладка Advanced (Дополнительно), перемикач Background services (Служб, работающих в фоновом режиме) у групі властивостей Processor Scheduling (Распределение времени процессора)). Довжина кванта фіксована й дорівнює 120 мс. Ця установка надає перевагу фоновим процесам.

Пошук потоку для виконання

Для виконання новий потік вибирається, коли:

- минув квант часу для потоку (з використанням алгоритму пошуку готового потоку);

- потік перейшов у стан очікування події (потік сам віддає квант часу і дає команду планувальникові запустити алгоритм пошуку готового потоку);
- потік перейшов у стан готовності до виконання (використовують алгоритм розміщення готового потоку).

Планувальник підтримує спеціальну структуру даних – *список готових потоків* (dispatcher ready list). У цьому списку зберігається 31 елемент – по одному для кожного рівня пріоритету. З кожним елементом пов'язана черга готових потоків, всі потоки з однаковим пріоритетом перебувають у черзі, яка відповідає їхньому рівню пріоритету.

Під час виконання алгоритму пошуку готового потоку планувальник переглядає всі черги потоків, починаючи з черги найвищого пріоритету (31). Як тільки під час цього перегляду трапляється потік, його відразу вибирають для виконання. За допомогою цього алгоритму вибирають перший потік непустої черги з найвищим пріоритетом. Можна сказати, що в межах однієї черги використовують алгоритм кругового планування, якщо не враховувати динамічну корекцію пріоритетів, яку ми розглянемо далі.

Алгоритм розміщення готового потоку поміщає потік у список готових потоків. Спочатку перевіряють, чи не володіє потік вищим пріоритетом, ніж той, котрий виконується в цей момент. При цьому новий потік негайно починає виконуватися, а поточний поміщається у список готових потоків; у противному разі новий потік поміщається в чергу списку готових потоків, відповідну до його пріоритету. У початку кожної черги розташовані потоки, які були витиснені до того, як вони виконувалися впродовж хоча б одного кванта, всі інші потоки поміщаються в кінець черги.

Якщо подивитися на ситуацію з боку потоку, що виконується, то важливо знати, коли він може бути витиснений. Це трапляється коли:

- потік перейшов у стан очікування;
- минув квант часу потоку;
- потік із вищим пріоритетом перейшов у стан готовності до виконання;
- змінився пріоритет потоку або пріоритет іншого потоку.

Динамічна зміна пріоритету і кванта часу

Під час виконання потоків динамічний пріоритет і довжина кванта часу можуть бути скориговані ядром системи. Розрізняють два види такої динамічної зміни: *підтримка* (boosting) і *ослаблення* (decay).

Підтримка зводиться зазвичай до тимчасового підвищення пріоритету потоків. Коли потік переходить у стан готовності до виконання внаслідок настання події, на яку він очікував, виконують операцію підтримки.

- Під час завершення операції введення-виведення підвищення пріоритету залежить від типу операції. Наприклад, після виконання дискових операцій пріоритет збільшують на одиницю, після введення із клавіатури або обробки події від миші – на 6.
- Під час зміни стану синхронізаційного об'єкта (докладніше такі об'єкти будуть розглянуті пізніше) пріоритет потоку, який очікує цієї зміни, збільшують на одиницю.
- Вихід з будь-якого стану очікування для потоків інтерактивних застосувань призводить до підвищення пріоритету на 2, таке саме підвищення відбувається під час переходу в стан готовності потоків, пов'язаних із відображенням інтерфейсу користувача.
- Для запобігання голодуванню потоки, які не виконувалися упродовж досить тривалого часу, різко підвищують свій пріоритет (цей випадок розглянемо окремо).

Зазначимо, що внаслідок операцій підтримки динамічний пріоритет потоку не може перевищити значення 15 (максимально допустимого динамічного пріоритету). Якщо операція підтримки вимагає підвищення пріоритету до величини, вищої за це значення, пріоритет збільшують тільки до 15.

Підвищення пріоритету внаслідок підтримки дедалі слабшає. Після закінчення кожного кванта часу поточний пріоритет потоку зменшують на одиницю, поки він не дійде до базового, після чого пріоритет залишають на одному рівні до наступної операції підтримки.

Ще одним видом підтримки є зміна кванта часу для інтерактивних застосувань. Якщо під час налаштування системи задано використання квантів змінної довжини, можна вказати, що для інтерактивних застосувань довжина кванта буде збільшуватися (це називають підтримкою кванта для інтерактивних застосувань). Якщо така підтримка задана, то коли інтерактивне застосування захоплює фокус, всі його потоки отримують квант часу, який дорівнює значенню підтримки (дозволене одне з можливих значень кванта, наприклад, 40 або 60 мс).

З іншого боку, значення кванта може й зменшуватися (слабшати). Так, під час виконання будь-якої функції очікування довжина кванта зменшується на одиницю.

Для потоків із пріоритетом реального часу динамічна зміна пріоритету або довжини кванта ніколи не відбувається. Єдиний спосіб змінити пріоритет таких потоків – викликати відповідну функцію із прикладної програми.

Запобігання голодуванню

Якщо в системі постійно є потоки з високим пріоритетом, може виникати голодування потоків, пріоритет яких нижчий. Для того щоб уникнути голодування, спеціальний потік ядра один раз за секунду обходить чергу готових потоків у пошуках тих, які перебували у стані готовності досить довго (понад 3 с) і жодного разу не отримали шансу на виконання. Коли такий потік знайдено, то йому присвоюють пріоритет 15 (і він дістає змогу негайного виконання); крім того, довжину його кванта часу подвоюють. Після того, як два кванти часу минають, пріоритет потоку і його квант повертаються до вихідних значень.

Цей алгоритм не враховує причин голодування і не розрізняє потоків інтерактивних і фонових процесів.

Програмний інтерфейс планування

Розглянемо функції Win32 API, за допомогою яких можна працювати із класами пріоритетів процесів і відносних пріоритетів потоків.

Для зміни класу пріоритету процесу використовують функцію `SetPriorityClass()`, для визначення поточного класу пріоритету – функцію `GetPriorityClass()`:

```
BOOL SetPriorityClass(HANDLE ph, DWORD pclass);
DWORD GetPriorityClass(HANDLE ph);
```

Параметр `ph` визначає дескриптор процесу, для якого задають клас пріоритету, а параметр `pclass` – клас пріоритету. Можливі такі значення `pclass`:

- `REALTIME_PRIORITY_CLASS` (реального часу);
- `HIGH_PRIORITY_CLASS` (високий);
- `NORMAL_PRIORITY_CLASS` (нормальний);
- `IDLE_PRIORITY_CLASS` (невикористовуваний).

Розглянемо приклад використання цих функцій:

```
HANDLE curh = GetCurrentProcess();
// задати клас пріоритету для поточного процесу
SetPriorityClass(curh, IDLE_PRIORITY_CLASS);
// взнати поточне значення класу пріоритету
printf("Поточний клас пріоритету:%d\n", GetPriorityClass(curh));
```

Щоб задати відносний пріоритет потоку, використовують функцію `SetThreadPriority()`, а щоб задати його значення – `GetThreadPriority()`.

```
BOOL SetThreadPriority(HANDLE th, int priority);
int GetThreadPriority(HANDLE th);
```

Параметр `th` є дескриптором потоку, параметр `priority` (і повернуте `GetThreadPriority()`) може набувати одного з таких значень:

- `THREAD_PRIORITY_TIME_CRITICAL` (критичний за часом);
- `THREAD_PRIORITY_HIGHEST` (найвищий);
- `THREAD_PRIORITY_ABOVE_NORMAL` (вищий за нормальний);
- `THREAD_PRIORITY_NORMAL` (нормальний);
- `THREAD_PRIORITY_BELOW_NORMAL` (нижчий за нормальний);
- `THREAD_PRIORITY_LOWEST` (найнижчий);
- `THREAD_PRIORITY_IDLE` (невикористовуваний).

Розглянемо приклад використання цих функцій.

```
DWORD tid;
// створення потоку
HANDLE th = _beginthreadex(... CREATE_SUSPENDED, &tid);
// задавання пріоритету
SetThreadPriority(th, THREAD_PRIORITY_IDLE);
// поновлення виконання потоку
ResumeThread(th);
// визначення пріоритету
printf("Поточний пріоритет потоку: %d\n",
GetThreadPriority(th));
// закриття дескриптора потоку
CloseHandle(th);
```

У даному прикладі ми створюємо потік у призупиненому стані, задаємо його пріоритет, а потім поновлюємо його виконання. Для цього використана функція `ResumeThread()`, що параметром приймає дескриптор потоку.

Висновки

- Задача планування потоків зводиться до організації виконання кількох потоків на одному процесорі так, аби у користувачів виникало враження, що вони виконуються одночасно. Цілями планування є: мінімізація часу відгуку, максимізація пропускну здатності та справедливість. До основних стратегій планування належать витісняльна й невитісняльна багатозадачність. У сучасних ОС застосовують витісняльну багатозадачність, коли рішення про перемикання контексту потоку приймають у коді ядра системи, а не в коді потоку.
- Розрізняють довготермінове, середньотермінове й короткотермінове планування.

Найважливіший тут короткотерміновий планувальник, котрий використовують для прийняття рішення про те, який потік запустити на виконання в певний момент. До основних алгоритмів короткотермінового планування належать планування кругове і з пріоритетами.