6.3 ВИКОРИСТАННЯ ПОТОКІВ

Настав час пояснити, чому ж потоки такі потрібні. Необхідність потоків простіше продемонструвати на конкретному прикладі. Уявіть собі, що користувач пише книгу. З точки зору автора найпростіше зберігати книгу в одному файлі, щоб легше було шукати окремі розділи, виконувати глобальну заміну і тому подібне. З іншого боку, можна зберігати кожну главу в окремому файлі.

Тепер уявіть собі, що користувач захотів видалити на 1-ій сторінці документу, в якому 800 сторінок, одне речення, а потім вирішив виправити речення на 600 сторінці (наприклад, задавши пошук фрази, що зустрічається тільки на цій сторінці). Текстовому редакторові доведеться переформатувати увесь документ аж до 600 сторінки. Це може зайняти відносно багато часу.

У цьому випадку допоможуть потоки. Нехай редактор написаний у вигляді двохпотокової програми. Один потік взаємодіє з користувачем, а другий переформатовує документ у фоновому режимі. Як тільки речення на першій сторінці була видалено, інтерактивний потік дає команду фоновому потоку переформатувати увесь документ. У той час коли перший потік продовжує відстежувати і виконувати команди з клавіатури або миші, другий потік швидко переформатовує документ. Форматування може закінчитися раніше, ніж користувач захоче перейти до 600 сторінки, і тоді команда буде виконана миттєво.

Чому б у даному прикладі не додати ще один потік? Більшість текстових редакторів зберігають редагований текст один раз в декілька хвилин. Цим може займатися третій потік, не відволікаючи два, що залишилися.

Якби програма була однопоточною, тоді при кожній операції форматування або збереження файлу всі команди з клавіатури і миші ігнорувалися б до закінчення цих операцій. У користувача це створило б враження низької продуктивності. Програмний модуль з трьома потоками істотно простіший. Очевидно, що в цьому випадку модель з трьома процесами не буде невдалою, оскільки всім трьом необхідно працювати з одним і тим же документом. Три ж потоки спільно використовують загальну пам'ять, і все три мають доступ до документу.

Паралельні обчислення (а, отже, і ефективніше використання ресурсів центрального процесора, і менший сумарний час виконання задач) тепер уже часто реалізується на рівні потоків. Програма, яка оформлена у вигляді декількох потоків у рамках одного процесу, може бути виконана швидше за рахунок паралельного виконання її окремих частин. Наприклад, якщо електронна таблиця розроблена з урахуванням можливостей багатопотокової обробки, то користувач може запросити перерахунок робочого листа і одночасно продовжувати заповнювати таблицю або редагувати наступний документ.

Потреба в потоках виникла ще на однопроцесорних системах, оскільки вони дозволяють організувати обчислення ефективніше. Для багатопроцесорних систем потоки вже просто потрібні, оскільки вони дозволяють не лише реально прискорити виконання тих задач, які допускають їх природне розпаралелювання, але і завантажити процесор роботою, щоб він не простоював.

Унаслідок того, що потоки належать до одного процесу, виконуються в одному віртуальному адресному просторі, між ними легко організувати тісну взаємодію, на відміну від процесів, для яких потрібні спеціальні механізми обміну повідомленнями і даними. Ще одним аргументом на користь потоків є легкість їх створення і знищення, оскільки з потоком не пов'язані ніякі ресурси. На створення потоку йде приблизно в 100 разів менше часу, ніж на створення процесу.

Незважаючи на перераховані переваги потоків, використання потоків не є універсальним засобом розв'язання проблем паралелізму, і пов'язано з деякими утрудненнями.

1. Розробляти і відлагоджувати багатопотокові програми важче, ніж звичайні послідовні програми. Організація загального використання адресного простору декількома потоками вимагає, щоб програміст мав високу кваліфікацію.

2. Використання потоків може понизити продуктивність програми. Найчастіше це трапляється в однопроцесорних системах. Наприклад, спроба виконати складні розрахунки декількома потоками може зробити до зайвих витрат на перемикання між потоками, кількість виконуваних інструкцій залишається одна і та ж.

Переваги і недоліки використання потоків необхідно враховувати під час виконання будь-якого проекту. Наведемо декілька порад з використання потоків.

1. У разі використання однопроцесорної системи певна кількість паралельних потоків часто не прискорює роботу додатку, якщо кожен з потоків не вимагатиме частого введення-виведення, оскільки буде тільки додаткове навантаження на систему, витрачене на перемикання між потоками.

2. Потоки добре виконуються, коли вони незалежні. Але вони починають працювати непродуктивно, якщо вони змушені часто синхронізуватися для доступу до загальних ресурсів. Блокування і критичні секції не збільшують швидкість роботи системи.

3. Пам'ять віртуальна. Механізм віртуальної пам'яті стежить за тим, яка частина віртуального адресного простору повинна знаходитися в оперативній пам'яті, а яка має бути скинута у файл підкачування. Потоки ускладнюють ситуацію, якщо вони звертаються в один і той же час до різних адрес віртуального адресного простору додатка. Це значно збільшує навантаження на систему, особливо при невеликому об'ємі кеш-пам'яті.

4. Не слід покладати на потік декілька функцій. Чим простіше і менш багатозначна кожна з даних ситуацій, тим більше ймовірність того, що помилок вдасться уникнути.

5. Кожен раз, коли який-небудь з потоків намагається скористатися загальним ресурсом цього процесу, якому він належить, потрібно тим або іншим чином легалізувати і захистити свою діяльність. Хорошим засобом для цього є критичні секції, семафори і черги повідомлень.

Кожен потік виконується строго послідовно і має свій власний програмний лічильник і стек. Подібно до традиційних процесів (тобто процесів, що складаються з одного потоку), потоки можуть також породжувати потоки- нащадки, можуть переходити із стану в стан (виконання, очікування і готовність). Поки один потік заблокований, інший потік того ж процесу може виконуватися. Потоки ділять між собою процесор так, як це роблять процеси, відповідно до різних варіантів планування. Поки один потік заблокований (чи просто знаходиться в черзі готових до виконання задач), інший потік того ж процесу може виконуватися.

У різних ОС по-різному будуються стосунки між потоками-нащадками і їх батьками. Наприклад, в одних ОС виконання батьківського потоку синхронізується з

його нащадками, зокрема після завершення батьківського потоку ОС може знімати з виконання усіх його нащадків. В інших системах потоки-нащадки можуть виконуватися асинхронно стосовно батьківського потоку.