6.6 БАГАТОПОТОКОВЕ ПРОГРАМУВАННЯ

Ефективне використання апаратури, що підтримує паралелізм, і в першу чергу так званих SMP систем (Symmetric Multiprocessor Architectures – SMP; симетричної мультипроцесорної архітектури комп'ютерів), тобто машин, що мають декілька процесорів і загальну, тобто доступну кожному з них пам'ять, дозволило ширше впроваджувати механізм легковагових процесів, або потоків управління (threads).

На подібній апаратурі програма, що використовує потоки може виконувати більш ніж одну інструкцію одночасно. Це дозволить додатку основну частину часу роботи, яку займають обчислення, підвищити свою продуктивність на двопроцесорній машині практично в два рази.

Перш ніж перейти до конкретних переваг, які дає нам багатопотоквое програмування, необхідно пояснити значення деяких термінів які ми активно використовуватимемо.

Під **асинхронними** подіями ми розумітимемо події, які відбуваються незалежно (можливо одночасно) за винятком випадків, коли залежність встановлюється зовнішніми силами.

Терміном **конкуренція** описуватимемо ситуацію, коли здається, що процеси відбуваються одночасно, проте насправді вони можуть відбуватися послідовно. Цим терміном добре описується, наприклад, поведінка процесів, що одночасно виконуються на однопроцесорній машині, хоча нам і здається, що зараз виконується декілька процесів, але насправді в цей конкретний момент виконується тільки один процес, що отримав поточний квант процесорного часу. Терміном **паралелізм** описуватимемо ситуацію, коли два процеси виконуються одночасно, тобто, паралельно, не перетинаючись. Справжній паралелізм може проявлятися тільки на багатопроцесорних системах, тоді як **конкуренція** і на однопроцесорних і на багатопроцесорних системах. Іншими словами конкуренція є лише ілюзією паралелізму. А справжній паралелізм вимагає для одночасного виконання декількох процесів декількох виконавців.

Використовуючи потоки і конкуренцію, ми можемо підвищити продуктивність програм навіть без використання апаратури, що забезпечує істинний паралелізм.

Припустимо, що ми маємо два потоки в програмі, що виконується на однопроцесорній машині. Нехай перший потік починає довгу операцію введення/виведення, тоді другий потік отримує процесорний час. Таким чином, замість того щоб простоювати, як це було б у разі однопоточного варіанту, наша програма продовжує виконуватися.

У принципі того ж ефекту можна досягти і без потоків, використовуючи можливості асинхронного або неблокуючого введення-виведення. Перший спосіб грунтується на тому, що при спробі читання або запису, яка повинна заблокувати процес, операція приймається до виконання, але блокування не відбувається. Замість цього можна продовжувати обчислення або здійснювати введення-виведення іншими каналами. Операційна система сповістить нас про закінчення операції введення-виведення посиланням відповідного сигналу.

Другий же спосіб ґрунтується на тому, що час від часу за допомогою спеціального системного виклику додаток опитує файл, в який необхідно здійснювати уведення-виведення, – чи готові вони до виконання тієї або іншої операції. У разі готовності додаток може виконувати бажану операцію. Ясно, що в проміжках між опитуваннями додаток може виконувати якусь імовірно корисну роботу.

Проте обидва ці способи мають істотний недолік. Код, що виходить при їх реалізації, досить складний, оскільки в ньому доводиться розносити введення-виведення і обробку даних.

Особливу гостроту проблема введення-виведення набуває в мережевих додатках, де блокування однопотокового процесу повільною операцією введення-виведення в мережі при обслуговуванні одного із запитів веде до призупинення обслуговування інших запитів. Як правило, така поведінка для цих застосувань не прийнятна. Саме тому, найчастіше використовувалися можливості конкурентного виконання запитів, спочатку засновані на використанні багатьох процесів, потім на використанні багатьох потоків або неблокуючого введення-виведення.

Навіть якщо програма в багатопотоковому варіанті не працюватиме продуктивніше, то все одно можна отримати деякі переваги від використання потоків. Виділяючи в програмах незалежні події і послідовності подій, і оформляючи їх у

вигляді різних потоків, можна отримати програми, які краще відображають реальність, і як наслідок, які буде зручніше супроводжувати.

Природно, що за все хороше доводиться платити. Чим же нам доводиться платити в разі використання потоків?

По-перше, можливою втратою продуктивності. Очевидно, що навіть повністю розпаралелюваний обчислювальний додаток на однопроцесорній машині, у багатопотоковому варіанті виконуватиметься повільніше за свою однопоточну версію. Це обумовлено накладними витратами на управління потоками і забезпечення узгодження між ними (на синхронізацію). Далі, навіть при виконанні на багатопроцесорній машині в разі коду, що погано розпаралелений, ми можемо виявитися в ситуації, коли згадані накладні витрати перевищать виграш від використання паралелізму.

По-друге, написання багатопотокових програм вимагає більшої акуратності, ніж написання звичайних програм. До стандартних проблем, таким як вихід за межі своєї пам'яті, висячі посилання тощо додаються різноманітні проблеми, які пов'язані з паралельним програмуванням такі як тупіки, умови перегонів (race conditions) і багато інших (складність кодування).

По-третє, багатопотокові програми з цілого ряду причин важче відлагоджувати, чим однопоточні програми. Тут позначається і деяка недорозвиненість засобів відладки таких додатків, і те, що сам процес відладки, змінюючи часові характеристики виконання програми, може призвести до її поведінки, абсолютно відмінної від того, при якому проявлялася помилка.

І, нарешті, той простий факт, що якщо раніше, наприклад, зіпсувати пам'ять міг тільки сам процес, то зараз це може зробити будь-який з декількох процесів. У результаті найпотужнішим засобом відладки подібних програм, як і раніше, залишається мозок програміста.

Тільки уважно зважуючи усі перераховані і деякі інші чинники, слід приймати рішення про те, чи варто використовувати потоки в програмі, чи ні.