6.5 ПОТОКИ НА РІВНІ КОРИСТУВАЧА І НА РІВНІ ЯДРА

Виділяють дві загальні категорії потоків: потоки **на рівні користувача** (User-Level Threads – ULT) і **потоки на рівні ядра** (Kernel-Level Threads – KLT). Потоки другого типу називаються потоками, підтримуваними ядром або полегшеними (легковагими) процесами.

6.5.1 Потоки на рівні користувача (ULT)

У програмі, яка складається повністю з ULT-потоків, усі дії з управління потоками виконуються самими додатком, ядро і не підозрює про існування потоків. На рис. 6.5 показаний підхід, коли використовується тільки потоки на рівні (у просторі) користувача. Щоб додаток був багатопотоковим його потрібно створити з використанням спеціальної бібліотеки (система підтримки програм), яка є пакетом програм для роботи з потоками на рівні ядра.

Така бібліотека містить код, який дозволяє створювати і видаляти потоки, здійснювати обмін повідомленнями і даними між потоками, планувати їх виконання, а також зберігати і відновлювати їх контекст.

За умовчанням додаток на початку своєї роботи складається з одного потоку, і його виконання починається як виконання цього потоку. Такий додаток разом з його потоком розміщується в єдиному процесі, який управляється ядром.

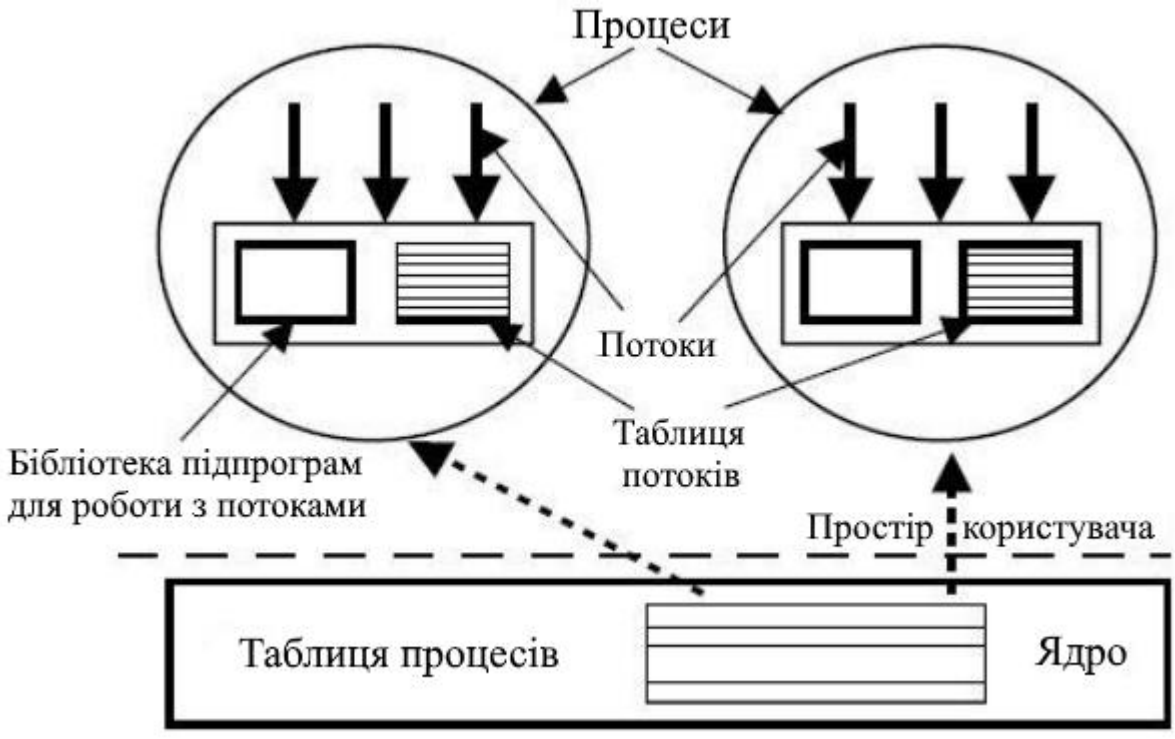

Рисунок 6.5 – Потоки в просторі користувача

Додаток, що виконується, у будь-який момент часу може створити новий потік, який виконуватиметься в межах того ж процесу. Цей новий потік створюється за допомогою виклику спеціальної підпрограми з бібліотеки, яка призначена для роботи з потоками. Управління до цієї підпрограми переходить у результаті виклику процедури. Бібліотека потоків створює структуру даних для нового потоку, а потім передає управління одному з готових до виконання потоків цього процесу відповідно до деякого алгоритму планування.

Коли управління переходить до бібліотечної підпрограми (система підтримки виконання програм), контекст поточного потоку зберігається, а коли управління повертається до потоку, його контекст відновлюється. Цей контекст (таблиця потоків), необхідний для відстежування потоків у процесі, в основному складається з вмісту регістрів користувача, лічильника команд і покажчиків стека. Кожному процесу потрібна таблиця потоків для відстежування потоків у процесі. Вона аналогічна таблиці процесів, з тією лише різницею, що вона відстежує лише характеристики потоків, такі як лічильник команд, покажчик вершини стека, регістри, стан тощо.

При цьому, усі описані раніше події відбуваються в призначеному для користувача просторі в рамках одного процесу. Ядро не підозрює про існування потоків. Воно продовжує здійснювати планування процесу як єдиного цілого і приписувати йому єдиний стан виконання (стан готовності, стан виконання, стан блокування тощо).

Використання потоків на призначеному для користувача рівні має деякі **переваги** перед використанням потоків на рівні ядра.

1. Перемикання потоків не включає перехід в режим ядра, оскільки структури даних по управлінню потоками знаходяться в адресному просторі одного і того ж процесу. Тому для управління потоками процесу не треба перемикатися в режим ядра. Завдяки цій обставині вдається уникнути накладних витрат, які пов'язані з двома перемиканнями режимів (режиму користувача в режим ядра і назад).

2. Планування здійснюється залежно від специфіки додатку. Для одних додатків кращим може бути простий алгоритм планування за круговим алгоритмом, а для інших – алгоритм планування на використанні пріоритету.

3. Використання потоків на призначеному для користувача рівні застосовне для будь-якої ОС. Для їх підтримки в ядро системи не потрібно вносити ніяких змін. Бібліотека потоків є набором утиліт, що працюють на рівні додатку і спільно використовуються усіма додатками.

Використання потоків на призначеному для користувача рівні має два явні **недоліки** в порівнянні з використанням потоків на рівні ядра:

1. У типовій ОС більшість системних викликів є блокуючими. Коли в потоці, який працює в режимі користувача, виконується системний виклик, то блокується не лише цей потік, але й усі потоки того процесу, до якого він належить.

2. У стратегії з наявністю потоків тільки на призначеному для користувача рівні додаток не може скористатися перевагами багатопроцесорної системи, оскільки ядро закріплює за кожним процесом тільки один процесор. Тому декілька потоків одного і того ж процесу не можуть виконуватися одночасно. По суті, у нас виходить багатозадачність на рівні додатку в рамках одного процесу.

Ці дві проблеми вирішувані. Наприклад, їх можна здолати, якщо писати додаток не у вигляді декількох потоків, а у вигляді декількох процесів. Але при такому підході основні переваги потоків зводяться нанівець: кожне перемикання стає не перемиканням потоків, а перемиканням процесів, що призводить до значних накладних витрат.

Іншим методом подолання проблеми блокування є використання перетворення блокуючого системного виклику в неблокуючий. Наприклад, замість безпосереднього виклику системної процедури введення-виведення потік викличе програмну оболонку, яка здійснює уведення-виведення на рівні додатку. У цій програмі міститься код, який перевіряє, чи зайнятий пристрій введення-виведення. Якщо він зайнятий, то потік передає управління іншому потоку (за допомогою бібліотеки потоків). Коли потік знову отримує управління, він повторно здійснює перевірку зайнятості пристрою введення-виведення. Але подолання усіх цих проблем виливається в складність їх реалізації.

6.5.2 Потоки на рівні ядра (KLT)

Тепер розглянемо ситуацію, в якій ядро знає про існування потоків і управляє ними. У програмах, робота яких повністю заснована на потоках, що працюють на рівні ядра, усі дії з управління потоками виконуються ядром. В області додатків відсутній код, який призначений для управління потоками. Немає необхідності і в наявності таблиці потоків у кожному процесі, замість цього є єдина таблиця потоків, що відстежує всі потоки системи. Замість коду управління потоками в процесі використовуються інтерфейс прикладного програмування (API) засобів ядра, що управляють потоками. Прикладами такого підходу є ОС OS/2, LINUX і W2K.

На рис. 6.6 показана стратегія використання потоків на рівні ядра. Будь- який додаток при цьому можна запрограмувати як багатопотоковий; усі потоки додатка підтримуються в рамках єдиного процесу. Ядро підтримує інформацію контексту процесу як єдиного цілого, а також контекстів кожного окремого потоку процесу.

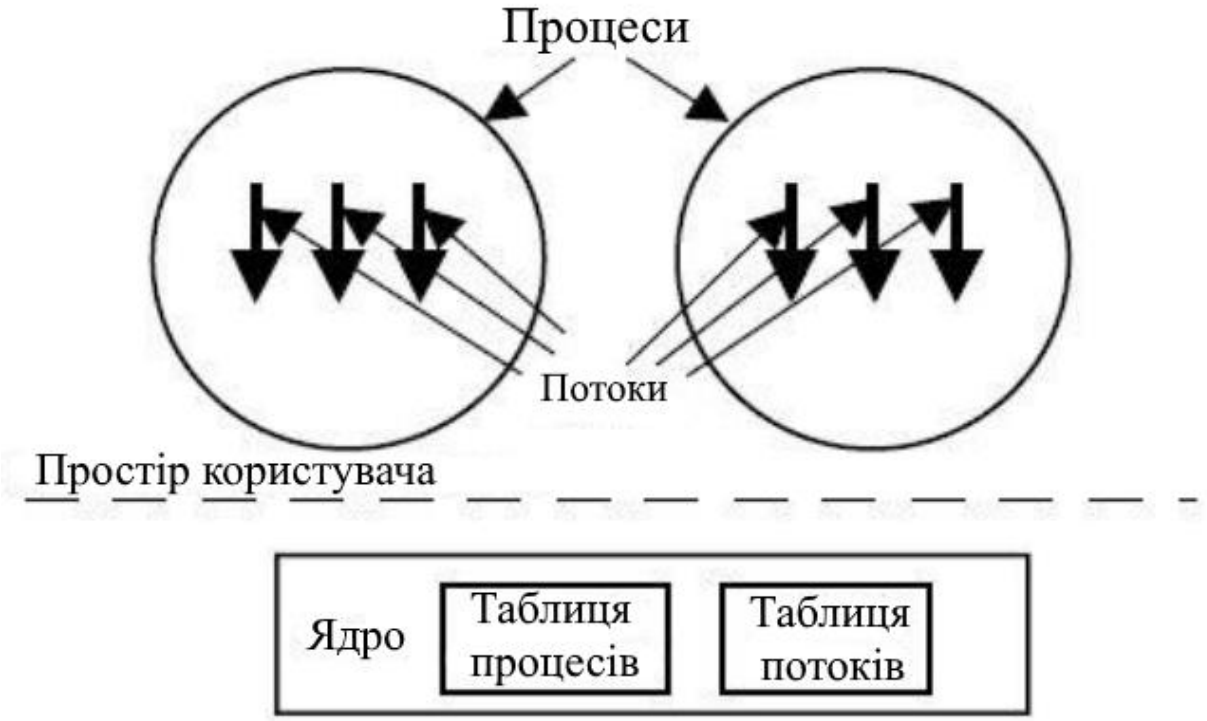

Рисунок 6.6 – Потоки в просторі ядра

Планування виконується ядром виходячи зі стану потоків. За допомогою такого підходу вдається позбавитися від двох згаданих раніше недоліків потоків рівня користувача.

По-перше, ядро може одночасно здійснювати планування роботи декількох потоків одного і того ж процесу на декількох процесорах. По-друге, при блокуванні

одного із потоків процесу ядро може вибрати для виконання інший потік цього ж процесу. Ще однією перевагою такого підходу є те, що самі процедури ядра можуть бути багатопоточними.

Основним недоліком підходу з використанням потоків на рівні ядра, в порівнянні з використанням потоків на рівні користувача, є те, що для передачі управління від одного потоку до іншого в рамках одного і того ж процесу доводитися перемикатися в режим ядра.

6.5.3 Комбіновані підходи

У деяких ОС використовується комбінування потоків обох видів (Solaris). Декілька потоків на рівні користувача, що входять до складу додатку, відображаються в таку ж або меншу кількість потоків на рівні ядра. Програміст може змінювати число потоків на рівні ядра, підбираючи його таким, яке дозволяє досягти найкращих результатів.

При комбінованому підході кілька потоків одного і того ж додатку можуть одночасно виконуватися на декількох процесорах, а блокуючі системні виклики не призведуть до блокування всього процесу. При належній реалізації такий підхід буде поєднувати в собі переваги підходів, в яких використовується тільки потоки на рівні користувача або тільки потоки на рівні ядра, зводячи недоліки кожного з цих підходів до мінімуму.

6.5.4 Спливаючі потоки

Потоки часто використовуються в розподілених системах. Комп'ютери в розподілених системах взаємодіють, обмінюючись один з одним повідомленнями. Важливим прикладом може служити обробка вхідних повідомлень, наприклад запитів на обслуговування. Традиційний підхід полягає в наявності процесу або потоку, який блокується за системним запитом, чекаючи повідомлення. Коли повідомлення прибуває, воно приймається і обробляється.

Можливий і інший підхід, при якому після прибуття повідомлення система створює новий потік для його обробки, який називають спливаючим.

Основною перевагою спливаючих потоків є їх «свіжість» (вони створюються заново) – у такого потоку немає історії: регістрів, стека і іншої інформації, яку потрібно відновлювати.

Створення спливаючих потоків у просторі ядра завжди швидше і простіше, ніж у просторі користувача. До того ж зі спливаючого потоку в просторі ядра простіше отримати доступ до всіх таблиць ядра і пристроїв введення-виведення. З іншого боку, наявність помилок в потоці, розташованому в просторі ядра, може завдати істотно більший збиток.