“Київський коледж зв’язку”

Циклова комісія Комп’ютерної інженерії

**Відповіді на контрольні**

**Запитання з лекції №4**

Виконали студенти

групи РПЗ-83а

Криворучко К.В.

Дунаєнко Я.В.

Перевірив викладач

Повхліб В.С.

Київ 2021

1. У потоку є лічильник команд, що відслідковує, яку чергову інструкцію потрібно виконувати. У нього є регістри, в яких містяться

поточні робочі змінні. У нього є стек з протоколом виконання, що містить по одному фрейму для кожної викликаної процедури, яка

ще не повернула управління. Хоча потік може бути виконаний в рамках якого-небудь процесу, сам потік і його процес є різними поняттями

і повинні розглядатися окремо. Процеси використовуються для групування ресурсів в єдине ціле, а потоки є «сутністю», що розподіляється

для виконання на центральному процесорі.

Потоки додають до моделі процесу можливість реалізації декількох, у великій мірі незалежних один від одного, виконуваних завдань в

єдиному середовищі процесу. Потоки використовують єдиний адресний простір, але різні ресурси.

У всіх потоків процесу один адресний простір, отже, вони також спільно використовують одні й ті ж глобальні змінні та деякі інші

елементи притаманні процесу.

Подібно до традиційного процесу (тобто процесу тільки з одним потоком), потік повинен бути в одному з наступних станів: виконуваний,

заблокований, готовий чи завершений.

Слід врахувати, що кожен потік має свій власний стек. Стек кожного потоку містить по одному фрейму для кожної

уже викликаної, але ще не завершеної процедури. Такий фрейм містить локальні змінні процедури та адресу повернення управління по

завершенні її виклику. Кожен потік буде, як правило, викликати різні процедури тому кожному потоку потрібен свій власний стек.

1. POSIX Threads (pthreads)

\* IEEE's POSIX Threads Model:

\* Модель програмування потоків для UNIX platform

\* pthreads включає міжнародні стандарти ISO / IEC9945-1

\* Програмна модель pthreads визначає:

\* Створення потоків

\* Управління виконанням потоків

\* Управління розділяються ресурсами процесу

Стандарт народився як модель програмування потоків для Unix систем і ґрунтувався на стандарті IEC9945-1

Стандарт визначає інтерфейси

по створенню і управлінню потоками, за механізмами розграничування доступу до розділених ресурсів процесу.

\* main thread:

\* Створити коли функція main () (in C) або PROGRAM (in fortran) визиваеться загрузчиком процесу

\* Функція main () може створювати дочірні threads

\* Якщо основний потік завершує роботу, процес переривається навіть якщо всередині процесу існують інші потоки, якщо тільки не вжито

спеціальні дії

\* Для уникнення переривання процесу можна використовувати pthread\_exit ()

У програмі завжди існує один потік, який створюються при завантаження функції main в Сі і PROGRAM в фортране.Далі основний потік

може створити дочірні потоки. При цьому якщо основний потік завершується, автоматично (якщо не вжито спеціальних дій)

завершуються і

дочірні потоки.Для того, щоб завершення дочірніх потоків не відбувалося, можна використовувати функцію виходу з (основного)

потоку

pthread\_exit ()

1. Якщо розмістити потоки в просторі користувача то ядро нічого не знатиме про існування потоків, воно керуватиме звичайними,

однопотоковими процесами.

Перша і найочевидніша перевага полягає в тому, що набір потоків на рівні користувача може бути реалізований в операційній системі,

яка не підтримує потоки.

Потоки запускаються поверх системи підтримки виконання програм (run-time system), яка являє собою набір процедур, керуючих потоками.

Чотири з них , це pthread\_create, pthread\_еxit, pthread\_joinі pthread\_yield, але зазвичай в наборі є ще й інші процедури.

Коли управління потоками здійснюється в просторі користувача, кожному процесу необхідно мати свою власну таблицю потоків, щоб відстежувати

потоки, наявні в цьому процесі. Ця таблиця є аналогом таблиці процесів, наявної в ядрі, за винятком того, що в ній містяться лише

властивості, що належать кожному потоку.

Таблиця потоків управляється системою підтримки виконання програм. Коли потік переводиться в стан готовності або блокується, інформація,

необхідна для відновлення його виконання, зберігається в таблиці потоків, точно так само як ядро зберігає інформацію про процесах в таблиці

процесів. Перемикання потоків, здійснене таким чином, принаймні на порядок, а може бути, і більше, швидше, ніж перехоплення управління

ядром, що є вагомим аргументом на користь набору потоків, реалізованого на рівні користувача.

Ще одною перевагою реалізації потоків на рівні користувача є можливість надати кожному процесу свої власні налаштування алгоритму планування.

1. При реалізації потоків в ядрі, ядро знає про існування потоків і керує ними.

У цьому випадку

:

\*Система підтримки виконання програм не потрібна.

\*Немає необхідності і в наявності таблиці потоків в кожному процесі, натомість є єдина таблиця потоків, що відслідковує всі потоки системи.

\*Якщо потоку необхідно створити новий потік або завершити наявний, він виконує запит ядра, який створює або завершує потік, вносячи зміни

в таблицю потоків.

Таблиця потоків, що знаходиться в ядрі містить регістри, стан і іншу інформацію про кожному потоці.

Інформація та ж, що і в разі керування

потоками на рівні користувача, тільки тепер вона розташовується не в просторі користувача (всередині системи підтримки виконання програм),

а в ядрі.

Ця інформація є підмножиною інформації, яку традиційне ядро зберігає про кожного зі своїх однопоточних процесів (тобто підмножиною

стану процесу). Додатково ядро містить звичайну таблицю процесів, що відстежує всі процеси системи.

Всі запити, які можуть блокувати потік,

реалізуються як системні запити, що вимагає значно більших витрат часу, ніж виклик процедури системи підтримки виконання програм.

Коли потік

блокується, ядро за бажанням запускає інший потік з цього ж процесу (якщо є потік в стані готовності) або потік з іншого процесу. При

управлінні потоками на рівні користувача система підтримки виконання програм запускає потоки з одного процесу, поки ядро не передати процесор

іншому процесу (або поки не закінчуються потоки, що знаходяться в стані готовності).

Оскільки створення і та завершення потоків в ядрі

вимагають відносно великих витрат, деякі системи використовують повторне використання потоків.

Після завершення потік позначається що

не функціонує, але в іншому його структура даних, що зберігається в ядрі, не зачіпається. Пізніше, коли потрібно створити новий потік,

реактивується відключений потік, що допомагає заощадити на деяких накладних витратах.

При управлінні потоками на рівні користувача повторне

використання потоків теж можливо, але оскільки накладних витрат, пов'язаних з управлінням потоками, в цьому випадку істотно менше, то і

сенсу в цьому менше.

Управління потоками в ядрі не вимагає нових не блокують системних викликів. Більш того, якщо один потік викликав помилку

через відсутність сторінки, ядро легко може перевірити, чи є в цьому процесі потоки в стані готовності, і запустити один з них, поки

потрібна сторінка зчитується з диска.

Основним недоліком управління потоками в ядрі є істотна ціна системних запитів

, Тому постійні помилки

з потоками (створення, завершення і т.п.) приведуть до збільшення накладних витрат.

1. У спробах об'єднати переваги створення потоків на рівні користувача і на рівні ядра була досліджена маса різних шляхів. Один з них полягає

у використанні потоків на рівні ядра, а потім декількох потоків на рівні користувача в рамках деяких або всіх потоків на рівні ядра. При

використанні такого підходу програміст може визначити, скільки потоків використовувати на рівні ядра і на скільки потоків розділити кожен

з них на рівні користувача. Ця модель має максимальну гнучкість. При такому підході ядру відомо тільки про потоках самого ядра, роботу яких

воно і планує. У деяких з цих потоків можуть бути кілька потоків на призначеному для користувача рівні, які розходяться від їх вершини.

Створення, видалення та планування виконання цих потоків здійснюється точно так же, як і у призначених для користувача потоків, що належать

процесу, запущеного під управлінням операційної системи, нездатною на многопоточную роботу. У цій моделі кожен потік на рівні ядра володіє

певним набором потоків на рівні користувача, які використовують його по черзі.