Тема 3

Багатопотоковість

1. Потоки: призначення, переваги, виклики

? Чим *потік* відрізняється від *процесу*?

[*Повторення*]

Потік (потік виконання, *thread*) - код програми, виконуваний на ЦП.

Узагальнене означення процесу:

> **Процес** (*process*) - один або декілька потоків виконання, а також сукупність пов’язаних з ними ресурсів.

Багатопотоковість: у процеса може бути один потік, а може бути й більше потоків.

Наприклад:

У цього процеса один потік

У цього процеса багато потоків

У деяких із цих процесів один потік, в інших - більше потоків

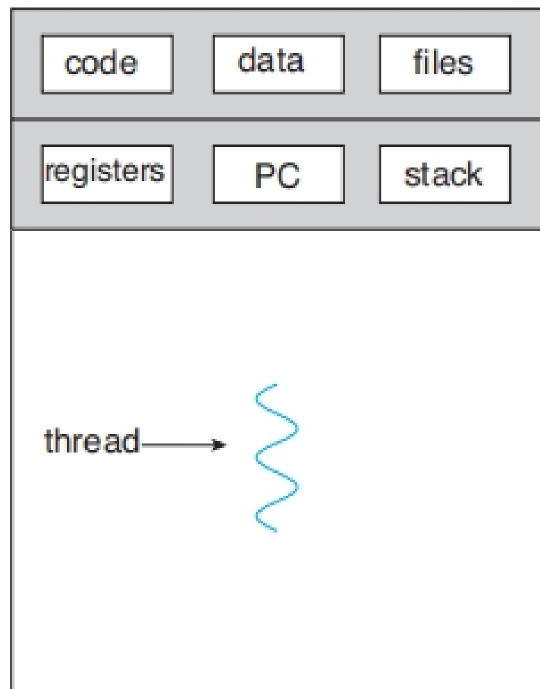
1. Потоки: призначення, переваги, виклики

З ***потоком*** пов'язані:

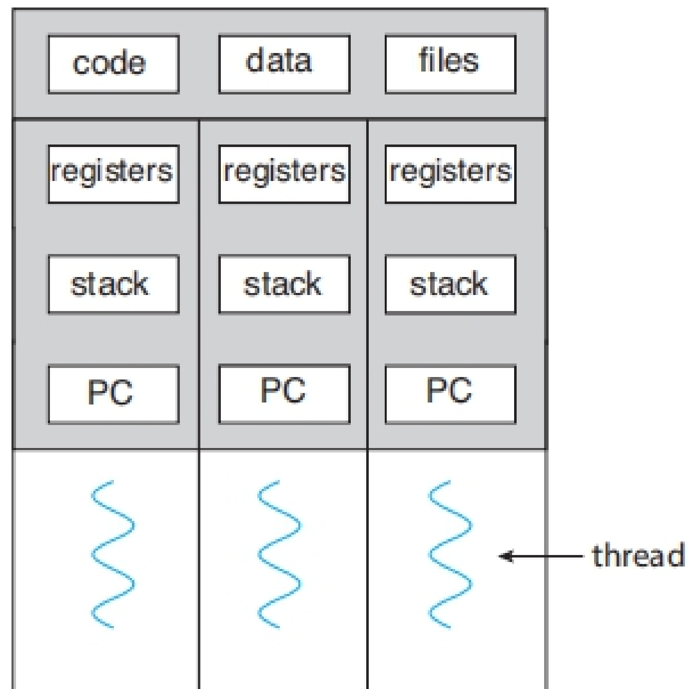
- лічильник команд (*program counter, PC*)
- значення інших регістрів ЦП
- стек

З ***процесом*** пов'язані:

- програмний код
- дані (котрі не в регістрах і не в стеку)
- інші ресурси
  (наприклад, відкриті файли, сигнали)



single-threaded process



multithreaded process

Джерело схеми: *A. Silberschatz, P. Galvin and G. Gagne,* Operating system concepts, 10th ed., Wiley, 2018. - p. 160, fig. 4.1

1. Потоки: призначення, переваги, виклики

? Навіщо нам потоки?

Переваги багатопотоковості

- **Спільне використання ресурсів**

  У потоків одного процесу спільний адресний простір.

- **Оперативність реакції програми *(app responsiveness)***

  Можна зробити окремий потік, відповідальний за взаємодію з користувачем. Основну роботу виконуватимуть інші.

- **Економія часу і пам'яті**

  Потоки створюються швидше, ніж процеси.
  Перемикання контекстів між потоками теж швидше (потенційно).

- **Масштабованість**

  Програма може адаптуватися під апаратні можливості:

  - більша кількість потоків - на більшій кількості ядер,
  - менша кількість потоків - на меншій кількості ядер.

1. Потоки: призначення, переваги, виклики

Чому багатозадачність і паралелізм - не цілком взаємозамінні терміни

Багатозадачність (*multitasking*, *concurrency*) можна реалізувати:

- лише через псевдопаралелізм (апаратного паралелізму немає)
- із використанням апаратного паралелізму (+ псевдо- теж може бути)

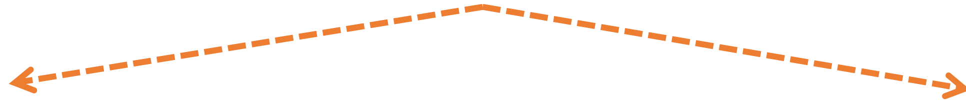
Паралелізм (*parallelism*), коли не має префікса “псевдо”, одними авторами тлумачиться як справжній, апаратний, паралелізм (багатоядерний процесор, багатопроцесорна система тощо), іншими авторами - як будь-який паралелізм, у т. ч. псевдопаралелізм.

Термін “паралелізм” треба використовувати, розуміючи ці нюанси.

Багатопотоковість розкриває свій потенціал, коли є не лише псевдопаралелізм, а й апаратний паралелізм.

1. Потоки: призначення, переваги, виклики

Багатопотоковість
ставить
виклики



перед розробниками ОС

як спланувати роботу у багатопотоковій системі?

(планування - *тема 4*)

перед прикладними програмістами

як написати багатопотокову програму?

Як виокремити всередині програми паралельні завдання? (і щоб кожне завдання було варте відокремлення)

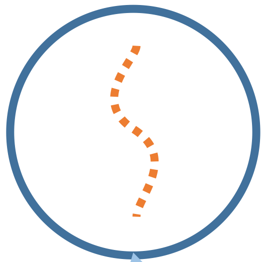
Як узгодити роботу паралельних завдань зі спільними даними?

Як усе це налагодити та протестувати? (послідовність виконання потоків може відрізнятися)

2. Основні моделі та стратегії багатопотоковості

Моделі процесів і потоків (залежно від їхньої кількості)

Один процес, один потік

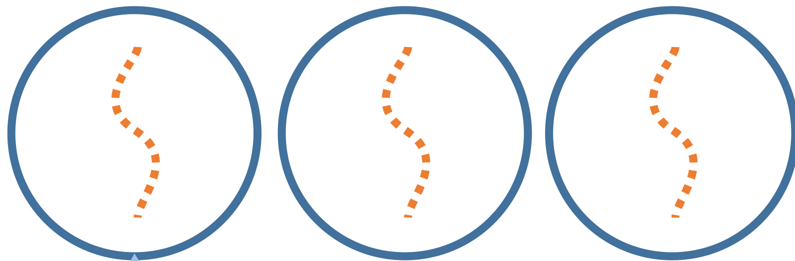


MS DOS

Один процес, багато потоків

деякі вбудовані ОС

Багато процесів (до появи сучасних потоків)



класичний UNIX

Багато процесів, багато потоків

більшість сучасних ОС

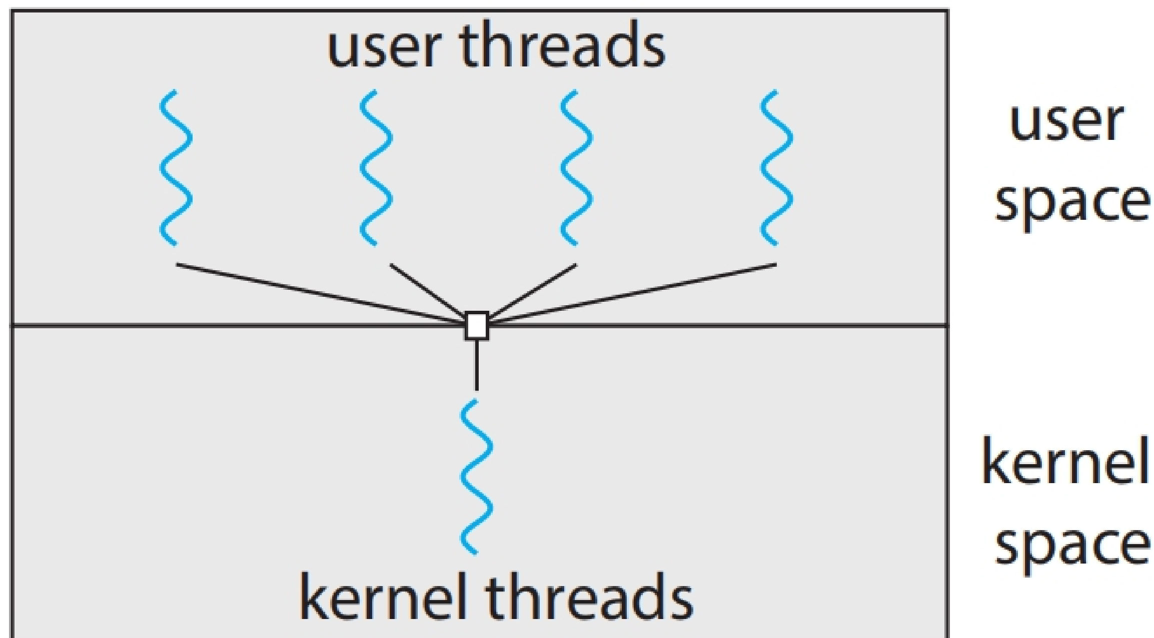
2. Основні моделі та стратегії багатопотоковості

Основні моделі багатопотоковості

- **Багато до одного** (***Many-to-one***)
- **Один до одного** (***One-to-one***)
- **Багато до багатьох** (***Many-to-many***)
- Інші моделі

2. Основні моделі та стратегії багатопотоковості

Модель “Багато до одного” *(Many-to-one)*



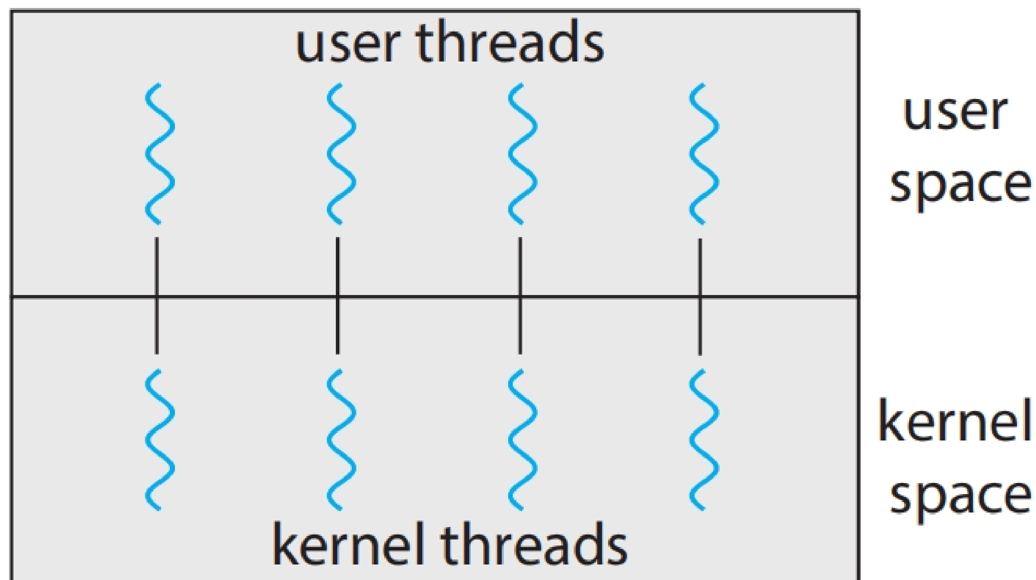
- Потоки реалізовано у просторі користувача.
- Ядро нічого не знає про потоки.
- Коли один з потоків процесу виконується, решта не може.

Приклади: рання Solaris, рання Java та ін.

Джерело схеми: *A. Silberschatz, P. Galvin and G. Gagne,* Operating system concepts, 10th ed., Wiley, 2018. - p. 166, fig. 4.7

2. Основні моделі та стратегії багатопотоковості

Модель “Один до одного” *(One-to-one)*



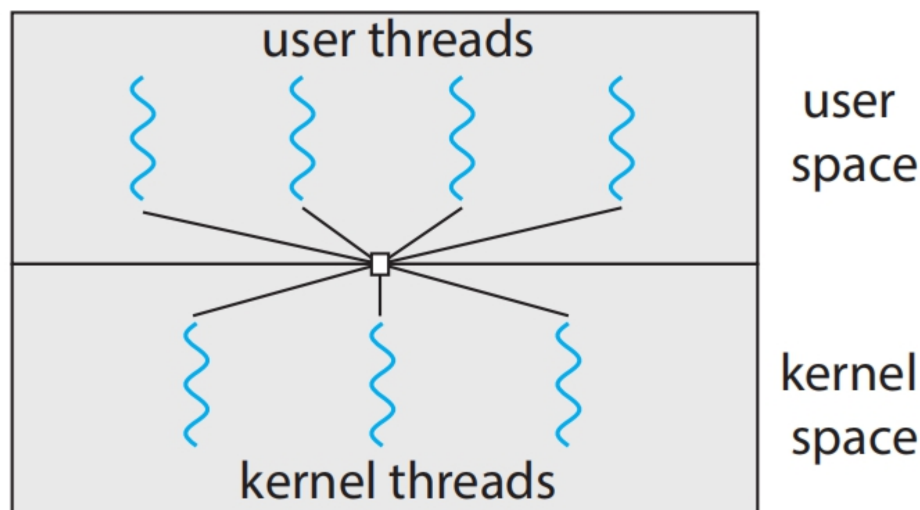
- Кожному потоку у просторі користувача відповідає потік у просторі ядра.
- Якщо процесор багатоядерний, потоки одного процесу зможуть виконуватися паралельно.
- Потоків ядра може стати забагато. Це погіршить роботу ОС.

Приклади: більшість сучасних ОС, зокрема Linux, Windows.

Джерело схеми: *A. Silberschatz, P. Galvin and G. Gagne,* Operating system concepts, 10th ed., Wiley, 2018. - p. 167, fig. 4.8

2. Основні моделі та стратегії багатопотоковості

Модель “Багато до багатьох” *(Many-to-many)*



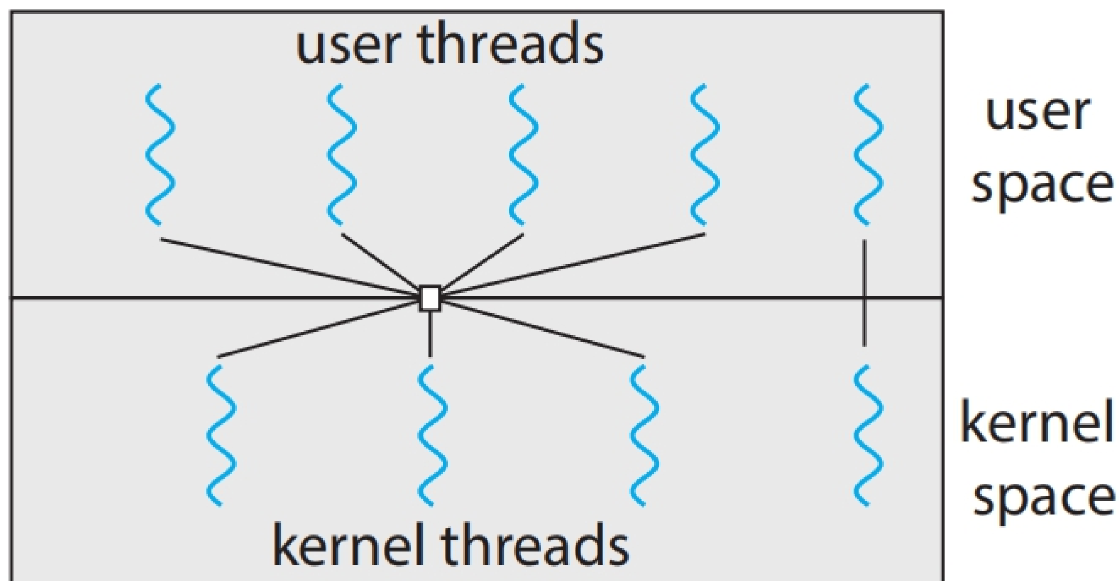
- Окремі потоки у просторі користувача, окремі - у просторі ядра.
- Кількість потоків ядра менша або рівна кількості потоків користувача.
- Менша кількість потоків ядра, ніж у моделі “Один до одного”.
- Важко реалізувати на практиці.

Приклади: на рівні бібліотек.

Джерело схеми: *A. Silberschatz, P. Galvin and G. Gagne,* Operating system concepts, 10th ed., Wiley, 2018. - p. 167, fig. 4.9

2. Основні моделі та стратегії багатопотоковості

Дворівнева модель *(Two-level model)*



- Частина потоків - як у моделі “Багато до багатьох”.
- Частина потоків - як у моделі “Один до одного”.

Приклади: HP-UX та ін.

Джерело схеми: *A. Silberschatz, P. Galvin and G. Gagne,* Operating system concepts, 10th ed., Wiley, 2018. - p. 168, fig. 4.10

2. Основні моделі та стратегії багатопотоковості

Основні стратегії багатопотоковості (на базі апаратного паралелізму)

Багатопотоковість завдань

Поділ даних між ядрами. Над кожним набором даних виконуються ті самі операції, але різними потоками.

Багатопотоковість даних

Кожний потік виконує різні операції.

Можуть поєднуватися

2. Основні моделі та стратегії багатопотоковості

Основні стратегії багатопотоковості (залежно від синхронності виконання)

Батьківський потік створив дочірній потік

Чи чекає батьківський потік на завершення дочірнього?

ні

так

Асинхронна багатопотоковість (asynchronious threading)

Батьківський потік продовжує виконання.

Використовується, коли потрібний користувацький інтерфейс з оперативною реакцією.

Синхронна багатопотоковість (synchronious threading)

Батьківський потік чекає на виконання дочірніх потоків, й лише тоді виконується далі.

Використовується, коли головне - спільна робота з даними (наприклад, паралельні обчислення).

3. Бібліотеки для роботи з потоками

Бібліотеки для роботи з потоками ← надають API

можуть бути реалізовані:
- на рівні користувача (потоки створюються без системних викликів)
- на рівні ядра (потоки створюються з системними викликами)

Приклади:

☆ **POSIX Pthreads** - реалізовані на рівні користувача або на рівні ядра

☆ **Windows Threads** - реалізована на рівні ядра

☆ **Java Threads** - рівень, на якому працюють потоки, залежить від реалізації потоків в основній ОС.

3. Бібліотеки для роботи з потоками

Проста багатопотокова програма на основі Pthreads (Ч. 1 з 3)

```
#include <iostream>
#include <pthread.h>
#include <math.h>
#include <stdlib.h>

using namespace std;

struct DATA_
{
 double x;
};

typedef struct DATA_ DATA;
double res1, res2;

void * F (void * arg);
void * G (void * arg);
```

3. Бібліотеки для роботи з потоками

Проста багатопотокова програма на основі Pthreads (Ч. 2 з 3)

```
int main()
{
 pthread_t threads[2]; // масив вказівників на потоки
 DATA arg;
 cout<<"Введіть значення аргумента x: "; cin>>arg.x;
 int er;
 er = pthread_create(&threads[0], NULL, F, (void *) & arg);
 if (er > 0)
 {
   cout<<"Не вдалося створити потік 1"<<endl;
   exit (EXIT_FAILURE);
 }
 er = pthread_create(&threads[1], NULL, G, (void *) & arg);
 if (er > 0)
 {
   cout<<"Не вдалося створити потік 2"<<endl;
   exit (EXIT_FAILURE);
 }
```

3. Бібліотеки для роботи з потоками

Проста багатопотокова програма на основі Pthreads (Ч. 3 з 3)

```
 for (int i = 0; i < 2; i++)
 {
   pthread_join (threads[i], NULL); // приєднуємо потоки
 }
 cout<<"F(x) + G(x) = "<<res1 + res2<<endl;
 return 0;
}
void * F (void * arg) // перша потокова функція
{
 DATA* a = (DATA*) arg;
 res1 = pow (a->x, 2);
 return 0;
}
void * G (void * arg) // друга потокова функція
{
 DATA* a = (DATA*) arg;
 res2 = pow (a->x, 3);
 return 0;
}
```

3. Бібліотеки для роботи з потоками

Проста багатопотокова програма на основі Windows Threads (ч. 1 з 3)

```
#include <iostream>
#include <windows.h>
#include <math.h>

using namespace std;

float res1, res2;
float x;

HANDLE hThreads[2]; // масив дескрипторів потоків
void F ();
void G ();

int main()
{
 cout<<"Введіть значення аргумента x: "; cin>>x;
 DWORD IDThread1, IDThread2; // змінні для ідентифікаторів потоків
```

3. Бібліотеки для роботи з потоками

Проста багатопотокова програма на основі Windows Threads (ч. 2 з 3)

```
hThreads[0] = CreateThread (NULL, 0, (LPTHREAD_START_ROUTINE)F,
                            NULL, 0, &IDThread1);
if (hThreads[0] == NULL)
{
  cout<<"Ne vdalosya stvoryty 1-i potik!"<<endl;
  return 1;
}

hThreads[1] = CreateThread (NULL, 0, (LPTHREAD_START_ROUTINE)G,
                            NULL, 0, &IDThread2);
if (hThreads[1] == NULL)
{
  cout<<"Ne vdalosya stvoryty 2-j potik!"<<endl;
  return 2;
}

// Чекаємо на завершення потоків так довго, як знадобиться
WaitForMultipleObjects (2, hThreads, TRUE, INFINITE);

CloseHandle (hThreads[0]);
CloseHandle (hThreads[1]);
```

3. Бібліотеки для роботи з потоками

Проста багатопотокова програма на основі Windows Threads (ч. 3 з 3)

```
 cout<<"f(x) + g(x) = "<<res1 + res2<<endl;
 return 0;
}

// Перша потокова функція
void F ()
{
 res1 = pow (x,2);
}

// Друга потокова функція
void G ()
{
 res2 = pow (x,3);
}
```

3. Бібліотеки для роботи з потоками

Опосередкована багатозадачність

Основні ідеї:

- паралельні завдання відокремлюються від реалізації багатопотоковості;
- розробник визначає паралельні завдання;
- бібліотеки відповідають за виконання цих завдань як потоків.

Приклади реалізації:

- **Пул потоків** (thread polls), використовується в Android, Java
- **Відгалуження-об'єднання** (fork-join), використовується в Unix-подібних ОС, Java)
- **OpenMP** (Open Multi-processing), ідея паралельних областей
- **Grand Central Dispatch** (Apple), використовується у macOS, iOS, ідея динамічного підлаштування кількості потоків у пулі;
- **Intel Thread Building Blocks** та ін.

Для самостійного читання

1. [***Silberschatz, Galvin, Gagne, 2018***] Chapter 4.
2. [***Stollings, 2017***] Chapter 4.
3. [***Tanenbaum, Bos, 2014***] Chapter 2 (paragr. 2.2).
4. [***Шеховцов, 2009***] Розділ 3.