**1. Дайте визначення поняттям процес, потік (нить). Поясніть як дані концепції пов’язані один з одним. Головні відмінності процесу та потоку виконання.**

процес є основною одиницею виконання в ОС.

потік є основною одиницею виконання на рівні процесу.

Процес є основної одиницею виконання в операційній системі. Кожен процес має свій адресний простір RAM, котрий є недоступний для інших процесів.

Кожен процес складається як мінімум з одного потоку виконання. Потоки виконання можуть створювати в процесі виконання програми.

Кожен потік виконання має власний стек, може мати власний набор виконуваних інструкцій, а також власний набір даних.

Потік виконання має доступ до всього адресного простору (кучі) свого батьківського процесу і можуть обмінюватися даними без створення спеціальних механізмів передачі даних.

**2. Особливості використання потоків в обраній студентом мові програмування, особливості передачі параметрів. Операції join та detach. Для чого необхідна операція detach.**

для створення потоку можна використовувати клас **std::thread** з стандартної бібліотеки.

**Передача параметрів**: Параметри можна передавати у функцію потоку через конструктор **std::thread**. Це може бути зроблено за допомогою передачі аргументів через конструктор або через захоплення змінних через лямбда-функції.

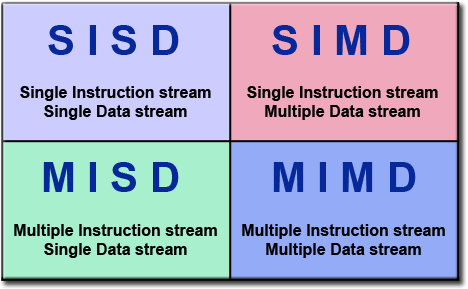
**Операції join та detach**:

* + **join()**: Ця операція чекає, поки потік завершить виконання своєї функції.
  + **detach()**: Ця операція відокремлює потік від основного потоку. Після виклику **detach()**, основний потік може продовжувати виконання незалежно від створеного потоку.

Операція **detach()** корисна в ситуаціях, коли основний потік не потрібно чекати на завершення роботи створеного потоку. Наприклад, коли основному потоку необхідно виконати іншу роботу або коли потік є фоновим і може завершити роботу після основної програми.

Отже, операція **detach()** відокремлює потік від основного потоку і дозволяє основному потоку продовжити виконання незалежно від створеного потоку.

**3. Перелічити категоризацію за Фліном. Пояснити кожну з окремих категорій та навести приклади пристроїв, що підпадають під відповідні категорії.**



* **SISD** (single instruction — single data) — одиночний потік команд і даних. Паралелізм відсутній.(IBM 360)
* **SIMD** (single instruction — multiple data) — одиночний потік команд і декілька потоків даних. Паралелізм в таких архітектурах полягає в можливості одночасного виконання однієї й тої ж операції над декількома елементами даних. Досягається це централізованою видачею команд декільком обчислювальним пристроям спільним для них пристроєм управління. Найпоширенішими представниками архітектур типу SIMD є так звані векторні ЕОМ, оптимізовані для паралельного виконання однотипних операцій над елементами векторів і матриць.(ILLIAC IV)
* **MISD** (multiple instruction — single data) — декілька потоків команд, одиночний потік даних. Фахівці дотепер не прийшли до єдиної думки щодо того, які архітектури відносити до даного класу. Деякі включають у нього конвеєрні обчислювачі, в яких цілісна операція розбивається на послідовність простіших етапів з суміщенням різних етапів для різних порцій даних у часі. Таким чином, в певному кожна порція даних від моменту завантаження з пам'яті до запису результату обробки проходить через декілька етапів обробки (стадій конвеєра). Подібні принципи покладені в основу систолічних архітектур, які також належать до цього класу.
* **MIMD** (multiple instructions — multiple data) — декілька потоків команд і даних. Найпоширеніший на сьогодні клас [паралельних архітектур](https://uk.wikipedia.org/w/index.php?title=%D0%9F%D0%B0%D1%80%D0%B0%D0%BB%D0%B5%D0%BB%D1%8C%D0%BD%D1%96_%D0%B0%D1%80%D1%85%D1%96%D1%82%D0%B5%D0%BA%D1%82%D1%83%D1%80%D0%B8_%D0%95%D0%9E%D0%9C&action=edit&redlink=1), у яких кожний процесор здатний працювати незалежно від інших над своїм завданням.(сучасні комп’ютери)

**4. Проблеми що виникають з CPU, RAM та кешом при використанні паралелізації. False-Positive sharing.**

**False-Positive Sharing**: Це ситуація, коли різні потоки виконання випадково отримують доступ до одного і того ж регістру кеша, що може призвести до зниження продуктивності через конфлікти доступу.

Паралелізація може призвести до різних проблем, пов'язаних з використанням процесора (CPU), оперативної пам'яті (RAM) та кеш-пам'яті. Деякі з найпоширеніших проблем включають:

**1. Перевантаження процесора (CPU):**

- При паралельному виконанні багатьох задач одночасно може виникнути конкуренція за ресурси процесора.

- Неправильне розподілення роботи між потоками може призвести до неефективного використання процесора, коли один потік чекає на інший або виникає багато маленьких задач, які затримують виконання.

- Проблеми з гонками при доступі до спільних ресурсів можуть затримати виконання потоків.

**2. Використання оперативної пам'яті (RAM):**

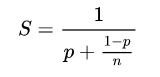
- Кожен потік вимагає свою власну частину пам'яті для виконання. Якщо потоків занадто багато або кожен потік вимагає велику кількість пам'яті, це може призвести до вичерпання оперативної пам'яті.

- Необхідно дбайливо керувати спільним доступом до об'єктів у пам'яті, щоб уникнути ситуацій гонок і некоректної роботи програми.

**3. Кеш-пам'ять:**

- При паралельному виконанні програми кожен потік може працювати зі своїм кешем. Це може призвести до проблеми false sharing, коли два або більше потоків змінюють різні об'єкти, що знаходяться в одному кеш-рядку. Це призводить до непотрібних копіювань даних між кешами і зниження продуктивності.

**5. Закон Амдала. Коефіцієнт прискорення та ефективності паралельного алгоритму.**

де �S — прискорення програми (як відношення до її початкового часу роботи); �p — частина яку можна виконувати послідовно; 1−�1-p — частина, яка виконується паралельно; �n  — кількість процесорів. Якщо послідовна частина програми виконується 10 % всього часу роботи, неможливо прискорити виконання такої програми більше ніж в 10 разів — незалежно від того, скільки процесорів використовує програма.

**6. Перелічити та описати типи паралелізму.**

Існує кілька типів паралелізму, які можуть бути використані для підвищення продуктивності програмного забезпечення:

**1. Паралелізм за даними (Data Parallelism):**

- У цьому типі паралелізму завдання розбиваються на декілька однотипних операцій, які виконуються над різними частинами даних.

- Наприклад, якщо ми маємо масив даних, можемо розділити цей масив на підмасиви та обробляти їх паралельно на різних процесорах або ядрах.

**2. Паралелізм за задачами (Task Parallelism):**

- У цьому випадку різні задачі розбиваються на незалежні підзадачі, які виконуються паралельно.

- Кожна підзадача може виконуватись на окремому потоці виконання або процесорі.