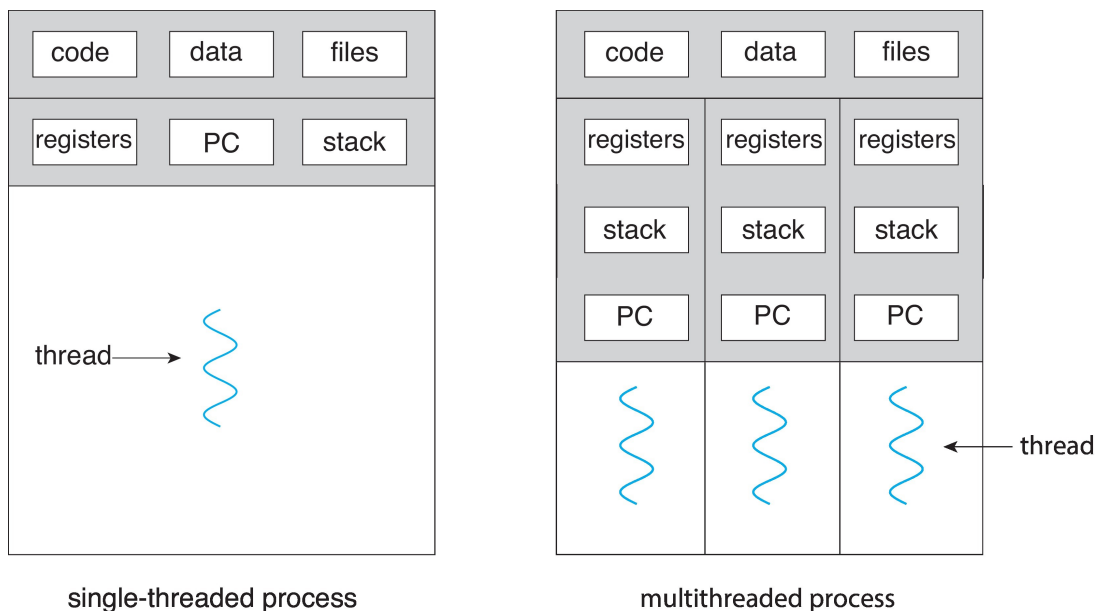


Threads

Threads

- A **thread** is a **single sequential flow of control within a program**.
- **Threads** are the **smallest unit of processing** that can be performed in an **operating system**.
- A **single process** can have **more than one thread**; each thread having a unique **program counter, stack, thread id, and set of registers**.
- **All threads** running under a **single process** have **shared memory (instructions, data, heap)**. They also can **communicate with each other directly**; there is no need for IPC.
- The **amount of time** it takes to **switch between threads** is less than the **amount of time** it takes to **switch between processes**.
- On **computer systems** that have **multiple cores**, **threads can run in parallel**.



Threads vs Processes

- **Threads** have **no data segment or heap**.
- A **process** has **code, data, heap, and stack segments**.
- A **thread cannot live on its own**; it needs to be attached to a process.
- **Processes** must have **at least one thread**.
- There can be **more than one thread in a single process**.
- **Threads** in a **process** share data.
- If a **thread dies**, its **stack is reclaimed**.
- If a **process dies**, all of its **threads die**.

POSIX Threads

- To **create a thread** you use the **pcreate_thread(thread, attr, start_routine, arg)** function.
 - **thread** — An opaque, unique identifier for the new thread returned by the subroutine.
 - **attr** — An opaque attribute object that may be used to set thread attributes.
 - **start_routine** — The routine that the thread will execute once it has been created.
 - **arg** — A single argument that may be passed to the **start_routine**.
- To **destroy a thread** you use the **pthread_exit(status)** function. This will **terminate the calling thread**, and **make the status available** to any **successful join with the terminating thread**.
- To **wait for a thread to terminate**, you use the **pthread_join(threadid, status)** function. This will **suspend the execution** of the **calling thread** until the **target thread terminates**.

User Threads and Kernel Threads

- **User threads** are **threads** that are **managed** at the **user-level**.
- **Kernel threads** are **threads** that are **managed by the kernel**.
- **User threads** and **kernel threads** have the same capabilities.

Multithreading Models

- There are three multithreading models:
 1. **Many-to-One**.
 2. **One-to-One**.
 3. **Many-to-Many**.