Multi-Threading

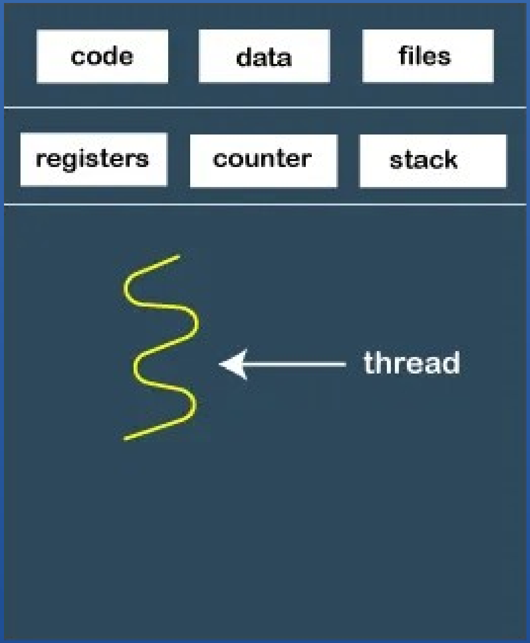
Asynchronous Programming

Definitions

- **Program**
  - A set of instructions written in a programming language (source code) that is translated into a compiled binary file or interpreted, enabling it to be executed by the computer.
- **Process**
  - An instance of a program that is being executed, including its code, data, and allocated system resources.
- **Thread**
  - The smallest unit of execution within a process that carries out tasks independently.
- **Multi-Thread**
  - The ability of a process to run multiple threads simultaneously to perform concurrent tasks.
- **Async Programming**
  - A programming model that enables **Tasks** to run independently without blocking the main thread, improving responsiveness.

- The study of **Operating Systems** is the discipline that explores foundational concepts like user interface, processes, threads, and resource management, as it deals with the design and functioning of systems that manage computer hardware and software.
- The Operating System (OS) is the responsible to mange the Process, Threads, and Tasks execution.

Process & Thread

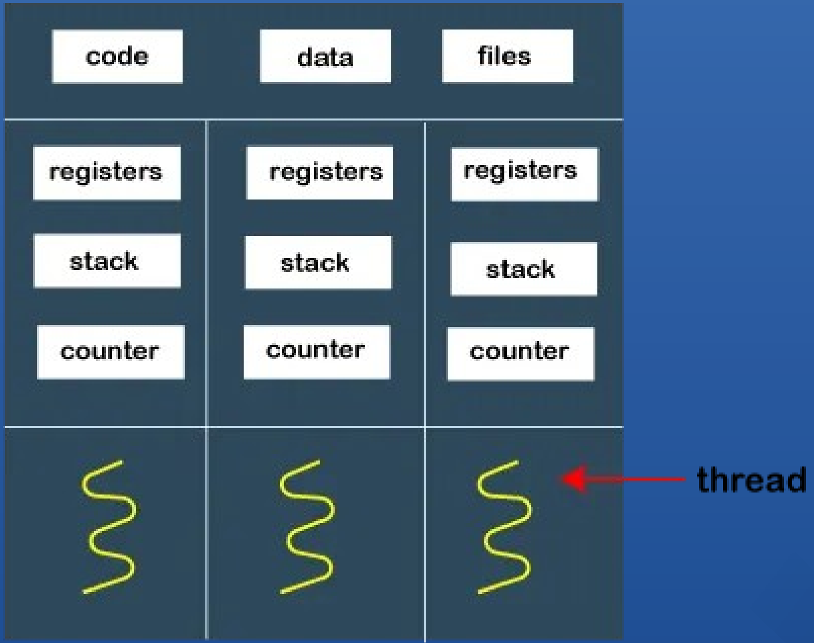


code data files

registers counter stack

thread

Single-threaded process



code data files

registers registers registers

stack stack stack

counter counter counter

thread

Multi-threaded process

Challenges and Pitfalls in Asynchronous Programming

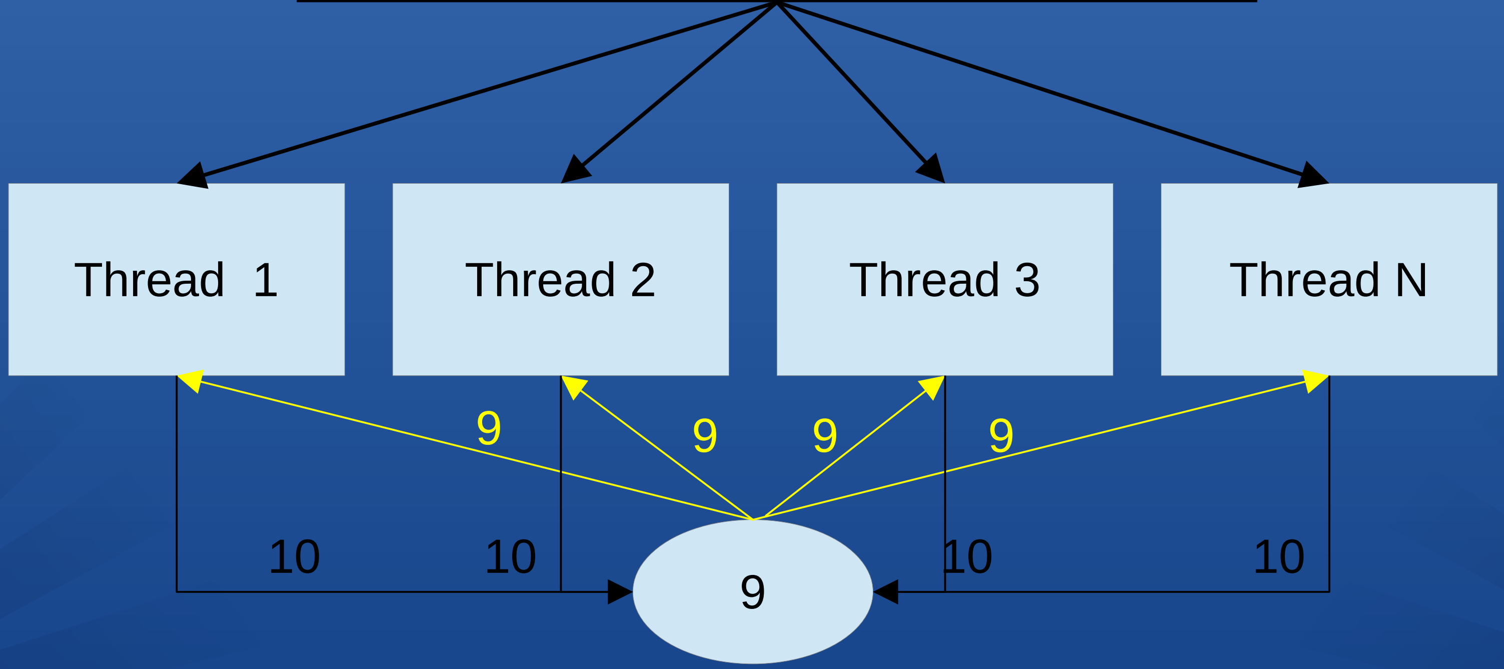
- **Race Conditions**
  - Occurs when multiple async operations access and modify shared data simultaneously.
  - Example: Two tasks checking and incrementing the same counter might produce inconsistent results.
- **Deadlocks**
  - Happens when two or more tasks wait indefinitely for each other to release a resource.
  - Example: Task A locks Resource X and waits for Resource Y, while Task B locks Resource Y and waits for Resource X.
- **Callback Hell**
  - Excessive nesting of callbacks makes code hard to read and maintain.
  - Example: Async function calling another async function, deeply nested.
- **Context Switching Overhead**
  - Frequent task switching can reduce performance instead of improving it.
  - Example: Overusing async tasks for CPU-bound operations can lead to inefficiency.
- **Debugging Complexity**
  - Tracing issues in async code is harder due to non-linear execution order.
  - Mitigation: Use structured logging, proper error handling, debugging tools.
- **Improper Exception Handling**
  - Unhandled exceptions in async tasks can be lost or cause unexpected behavior.
  - Mitigation: Always handle exceptions, use proper try/catch in async functions.

Race Condition

```
...
if (sharedVariable < 10) {
  DoSomething(….); // OS puts to sleep
  SharedVariable++;
} ...
```

Low level

```
temp = counter;
temp = temp + 1;
counter = temp;
```



Deadlock

Dinning Philosophers: Two forks required to eat