Name : Ibraheem

Sap id : 42896

Assignment no 4

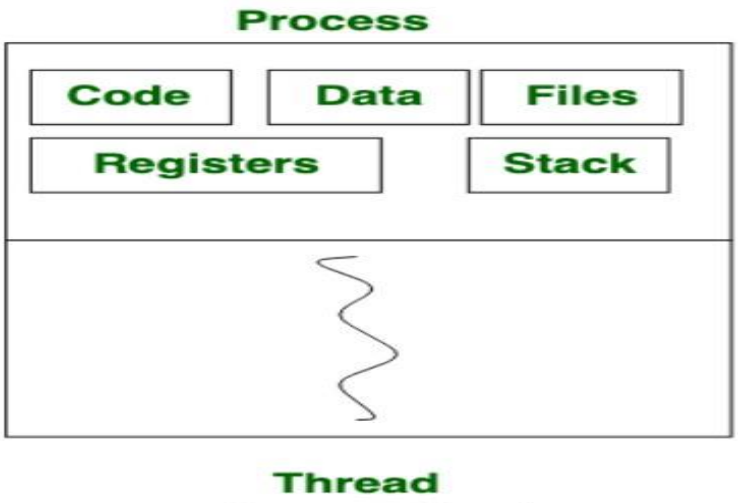
Q: what is threading? What are threads? What is the difference between threads and processes? **Answer:**

➢ What is Thread?

Threads are often called "lightweight processes" because they share some features of processes but are smaller and faster. Each thread is always part of one specific process. A thread has three states: Running, Ready and Blocked.
A thread takes less time to terminate as compared to the process but unlike the process, threads do not isolate.

➢ What is process?

A **process** is a program in execution, managed by the CPU. Its information is stored in a **Process Control Block (PCB)**. A process can create **child processes** and operates independently, not sharing memory with others. It may take longer to terminate and moves through states like **new, ready, running, waiting, suspended**, and **terminated**.



Process vs Thread

➢ Difference Between Process and Thread (Short Version):

Process	Thread

A program in execution	A part of a process
Slower to create and terminate	Faster to create and terminate
Heavyweight	Lightweight
Has its own memory	Shares memory with other threads
Slower context switching	Faster context switching
Uses separate PCB, stack, and address space	Shares address space; has own stack and TCB
Communication is less efficient	Communication is faster
One blocked process doesn't affect others	One blocked thread can block all threads in the process
Needs system calls to create	Created using APIs, no system call needed
Parent process changes don't affect child	Changes in one thread can affect others

Q: what is forking? Explain the usage of fork() and exec() in your own words along with few examples? **Answer :**

What is fork()**?**

- fork() is a **system call** used to create a **new process** by duplicating the current process.
- The original process is the **parent**, and the new one is the **child**.
- Both run independently with different process IDs.
- **Return Values:**
  - 0 → Child process
  - >0 → Parent process (PID of child) ○ -1 → Error

(no child created)

What is **exec()**?

- exec() is a **system call** that replaces the current process with a **new program**.
- It **does not create** a new process.
- Used **after fork()** if the child needs to run a different program. □ If successful, it never returns; if it fails, it returns -1.

➢ Example Summary (C Code)

- fork() creates a child.
- In the child, exec() runs the ls -lart /home command.
- Parent waits for the child to finish.

➢ fork() vs exec(): Short Table

Aspect	fork()	exec()
Type	System call in C	System call in OS
Function	Creates a new process	Replaces current process with new program
Return Type	Integer (PID or error)	Only returns on error (-1)
New Process Created	Yes	No
Code/Data Replacement	No	Yes – replaces code, data, heap, stack
Used For	Process duplication	Executing new programs
Parameters	No parameters	Takes program path and arguments
Control Returns To Code	Yes (both parent and child continue)	No (unless error)

Q: How threading is achieved in modern computers? Also describe various threading libraries?

Answer:

How Threading is Achieved in Modern Computers:

Modern computers achieve threading using **multicore processors** and **thread management** by the operating system. Each process can create multiple **threads** that share the same memory but run independently. The **OS scheduler** or **runtime environment** distributes threads across available CPU cores to enable **parallel** or **concurrent execution**.

Threading improves:

- CPU utilization
- Application responsiveness
- Performance in multitasking and background operations

Two main types of threads:

- **User-Level Threads (ULT):** Managed by the application, lightweight, but can't utilize multiple cores directly.
- **Kernel-Level Threads (KLT):** Managed by the OS, can run in parallel on multiple cores.

➢ Common Threading Libraries:

Library	Used In	Purpose
Pthreads (POSIX)	C/C++ (Unix/Linux)	Low-level thread creation and synchronization
std::thread	C++11 and above	Simple and safe thread creation in modern C++
Java Thread API	Java	Built-in support via Thread class and Runnable interface
Python threading	Python	Supports concurrent tasks; limited by GIL (Global Interpreter Lock)
OpenMP	C/C++/Fortran	Simplifies writing parallel code using compiler directives
.NET Task & Threading	C#/VB.NET	Multithreading with Thread, Task, and async/await
Boost.Thread	C++ (Boost)	Cross-platform, powerful threading support
Rust std::thread	Rust	Safe and efficient thread management with zero-cost abstractions