Centre Universitaire de Mila

2nd year of Computer Science degree (LMD)

Module : Operating Systems 1

Bessouf Hakim

CHAPTER 3 :

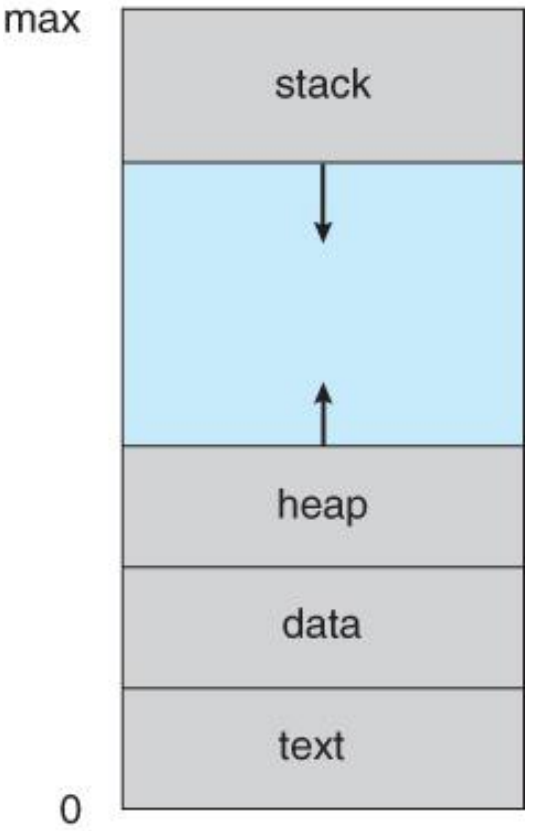
Processor Management

1. Definition of Scheduling / Scheduler
2. Objectives of Scheduling
3. Scheduling Criteria
4. Scheduling Levels (Job Scheduling, Process Scheduling)
5. Scheduling Policies
6. Process Control (Process State, Process Control Block - PCB, Process Creation, Termination, etc.)

Introduction

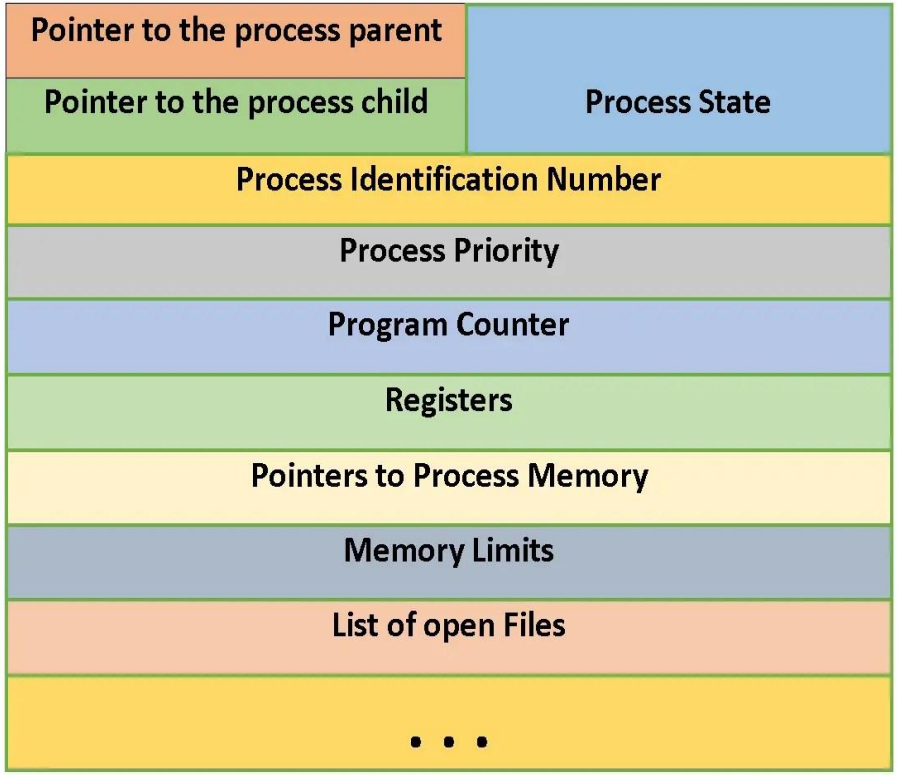
The memory space allocated to a process is called the Memory Map of the process. It is divided into several parts:

- **The Code**: This is the set of instructions of the program to be executed. This memory area is read-only.
- **The Data (Data Section)**: Contains all constants and declared variables.
- **The Stack**: Stores register values, local variables, function parameters, and return addresses.
- **The Heap**: A dynamically allocated memory area, managed during program execution using functions like new and malloc.



Process Management

- The operating system manages processes using a Process Table stored in central memory, known as the Process Control Block (PCB).
- This table contains information about each process, including:
  - Program Counter
  - Status Words
  - Stack Pointer
  - Allocated Resources (Memory, Files, etc.)



Process Manipulation

The operating system uses various primitives (procedures) to handle processes:

Creation: A process can create another process. The first is called the **parent** process, and the newly created one is the **child** process. A parent process can have **multiple** child processes.

Activation: Moves a ready process to the active state.

Suspension: Temporarily stops a process.

Termination: Ends a process and frees all associated resources.

Scheduling Criteria

- **Resource Availability:**

Programs are scheduled based on the resources they require and those available in the system.

- **Program Type:**

Programs are scheduled based on their category (interactive, real-time, computation-heavy, I/O-heavy, etc.).

- **Preemptive vs. Non-Preemptive Scheduling:**

In non-preemptive scheduling, a process runs until it is blocked (e.g., waiting for I/O) or it terminates.

In preemptive scheduling, the scheduler can suspend an active process to run another.

- **Priority Scheduling:**

Processes may have execution priorities (e.g., system processes get higher priority than user processes).

Scheduling Levels

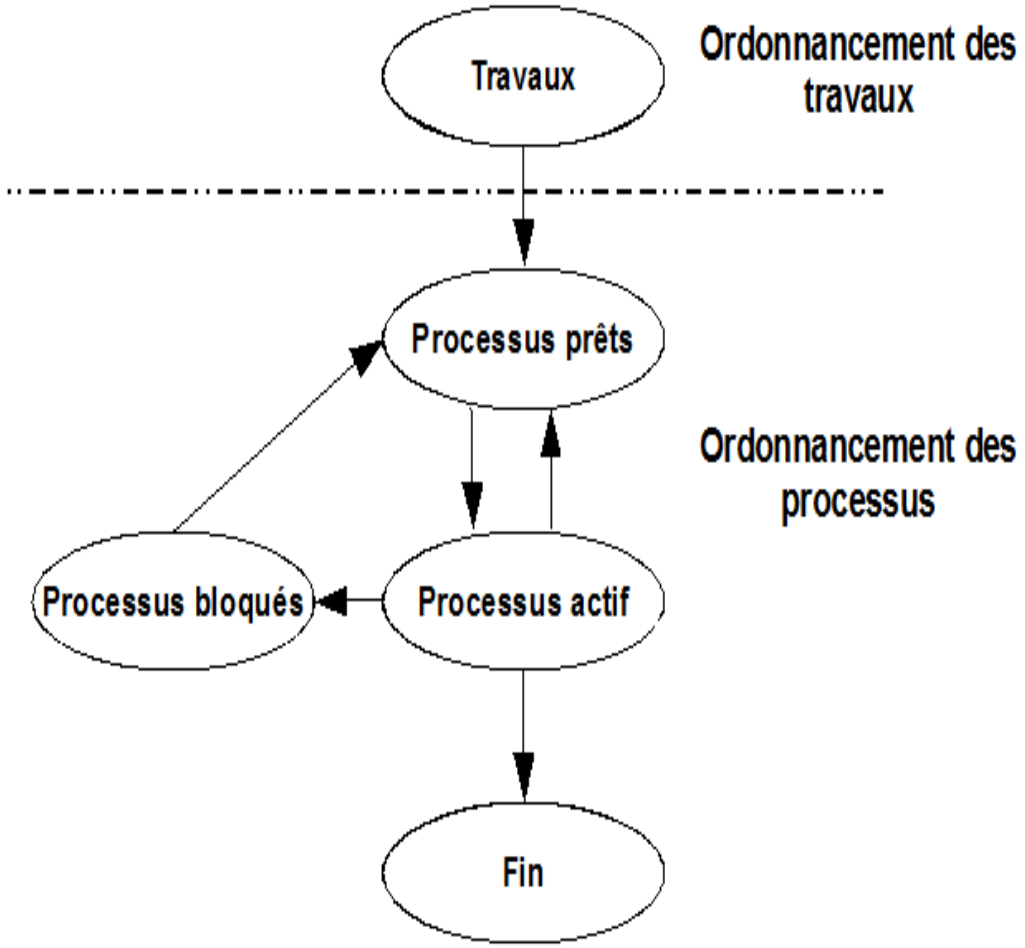
There are two main levels of scheduling:

Job Scheduling

Determines which job enters the system and creates the corresponding process.

Process Scheduling

Determines which process gets CPU time.



Job Scheduling (Long-Term Scheduling)

Memory Pool in Scheduling

- In the context of scheduling, a memory pool refers to a pre-allocated block of memory that is managed and used by the operating system to store processes that are waiting for execution. It plays a crucial role in job scheduling and process scheduling by ensuring that only processes with sufficient memory can be admitted into execution.

Job Scheduling (Long-Term Scheduling)

- When a job is submitted for execution, it is first placed in a job queue.
- The system checks if there is enough space in the memory pool before admitting the job into the ready queue.
- If the memory pool is full, the job must wait until memory becomes available.

Process Scheduling (Short-Term Scheduling)

Process Scheduling (Short-Term Scheduling)

- The memory pool contains all processes that are currently loaded into RAM and ready to execute.
- The CPU scheduler selects processes from this memory pool to assign CPU time.
- If a process requires more memory than available in the memory pool, it may be swapped out (paging in virtual memory) or delayed.

Process Scheduler

- The scheduler is a module of the operating system that distributes CPU time among multiple processes waiting for execution (in the ready state) based on predefined scheduling policies.
- The scheduler selects the next process to be executed, a decision known as scheduling.

Objectives of the Scheduler

The scheduler aims to:

- Ensure **fairness**:

It must allocate the CPU fairly among processes with the same priority.

- Maximize **resource utilization**:

The scheduler should ensure efficient use of system resources (CPU, memory, I/O).

Scheduler Objectives by System Type

- Execute as many jobs as possible per hour.
- Maximize CPU utilization.
- Minimize response time to user requests.
- Meet strict timing constraints.

Aspect	Job Scheduling	Process Scheduling
Definition	Selects jobs from the job pool to load into memory.	Selects processes from the ready queue to execute on the CPU.
Level	Higher level (job pool → memory).	Lower level (memory → CPU).
Frequency	Less frequent.	Very frequent.
Objective	Balance system load and resource utilization.	Maximize CPU utilization and fairness.
Example	Admitting a new job into memory.	Allocating CPU time to a process.
Impact on Performance	Affects system throughput and resource utilization.	Affects responsiveness and fairness.

Job Scheduling Policies

1- Non-Preemptive Scheduling Policies

First-Come, First-Served (FCFS) / FIFO:

The first job that arrives is executed first.

Disadvantage: Short processes may wait too long.

Shortest Job First (SJF):

The shortest job is executed first.

Advantage: Prioritizes shorter tasks.

Requirement: The system must predict execution times in advance.

Priority-Based Scheduling:

The scheduler executes the job with the highest priority first.

Priority may be preemptive or non-preemptive.

Job Scheduling Policies

2- Preemptive Scheduling Policies
Shortest Remaining Time First (SRTF):

The job with the shortest remaining execution time is scheduled first.

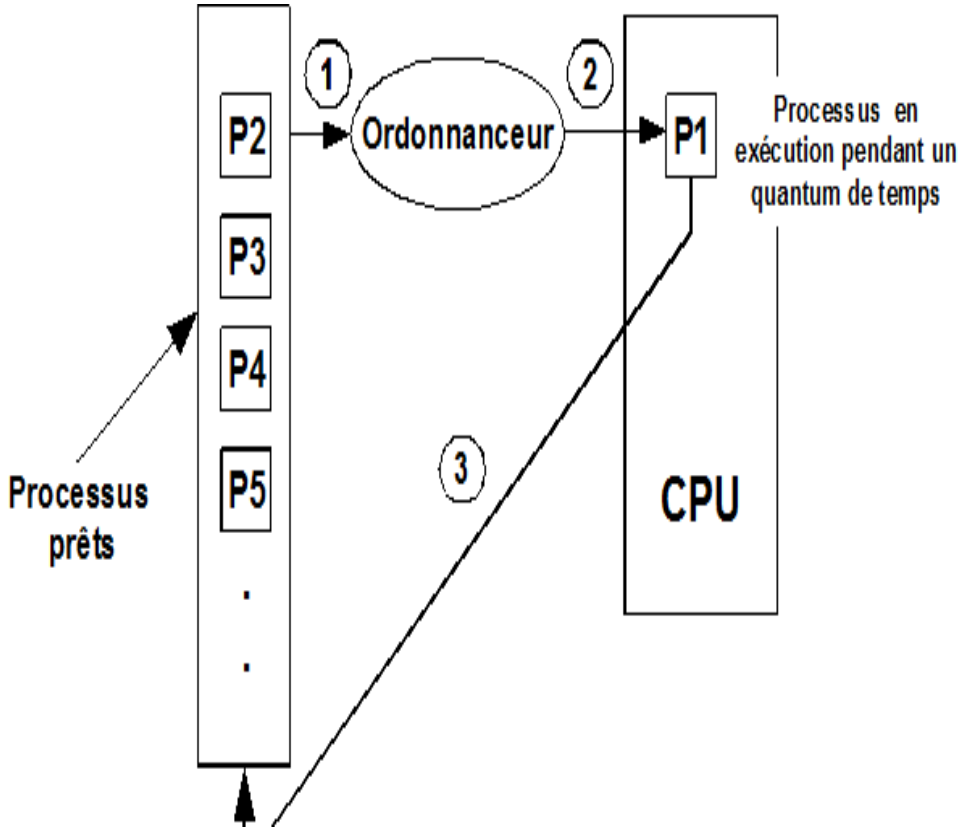
If a new job arrives with a shorter remaining time, the current job is preempted.

"This policy favors short jobs, but it requires knowing the execution time of the jobs in advance."

Process Scheduling Policies

Round Robin Scheduling

- Used in time-sharing systems.
- Similar to FCFS, but with time slices (quantum).
- Each process gets a fixed amount of CPU time before the next process runs.
- If the quantum is too large → behaves like FCFS.
- If the quantum is too small → increases CPU overhead due to frequent context switching.



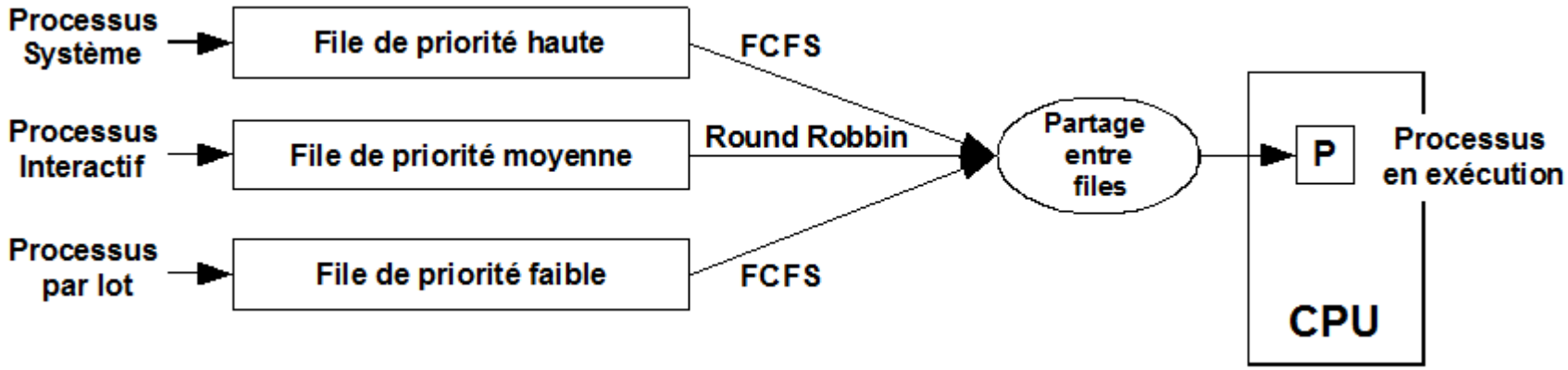
Process Scheduling Policies

Multi-Level Queue Scheduling

- A CPU scheduling algorithm that **divides the ready queue into multiple separate queues**, each designed for a specific type of process. Each queue may use a **different scheduling algorithm** and have its own priority
- Each queue has its own scheduling policy.
- A higher-level scheduler manages queue execution.
- The OS assigns priorities to queues.
- A higher-priority queue preempts lower-priority queues (if preemptive MLQ is used).
- If non-preemptive, a lower-priority queue only runs when higher-priority queues are empty.

Process Scheduling Policies

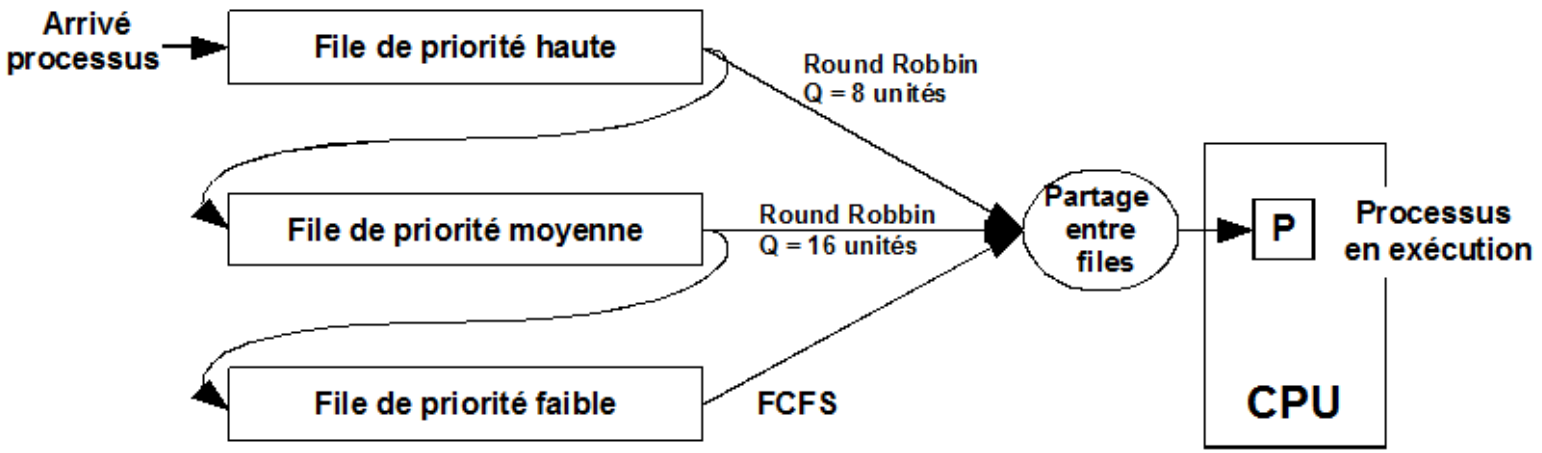
- **Multi-Level Queue Scheduling**
- **Windows Task Scheduling:**
  - **Real-time processes** (e.g., system services) have higher priority.
  - **Interactive applications** (e.g., MS Word, Chrome) run with time-sharing (Round Robin).
  - **Background tasks** (e.g., Windows Update, indexing) get CPU only when resources are available.



Process Scheduling Policies

Multi-Level Feedback Queue Scheduling

- Processes can change queues based on their execution behavior.
- Example: A CPU-bound process may be moved to a lower-priority queue over time.



Process Scheduling Policies

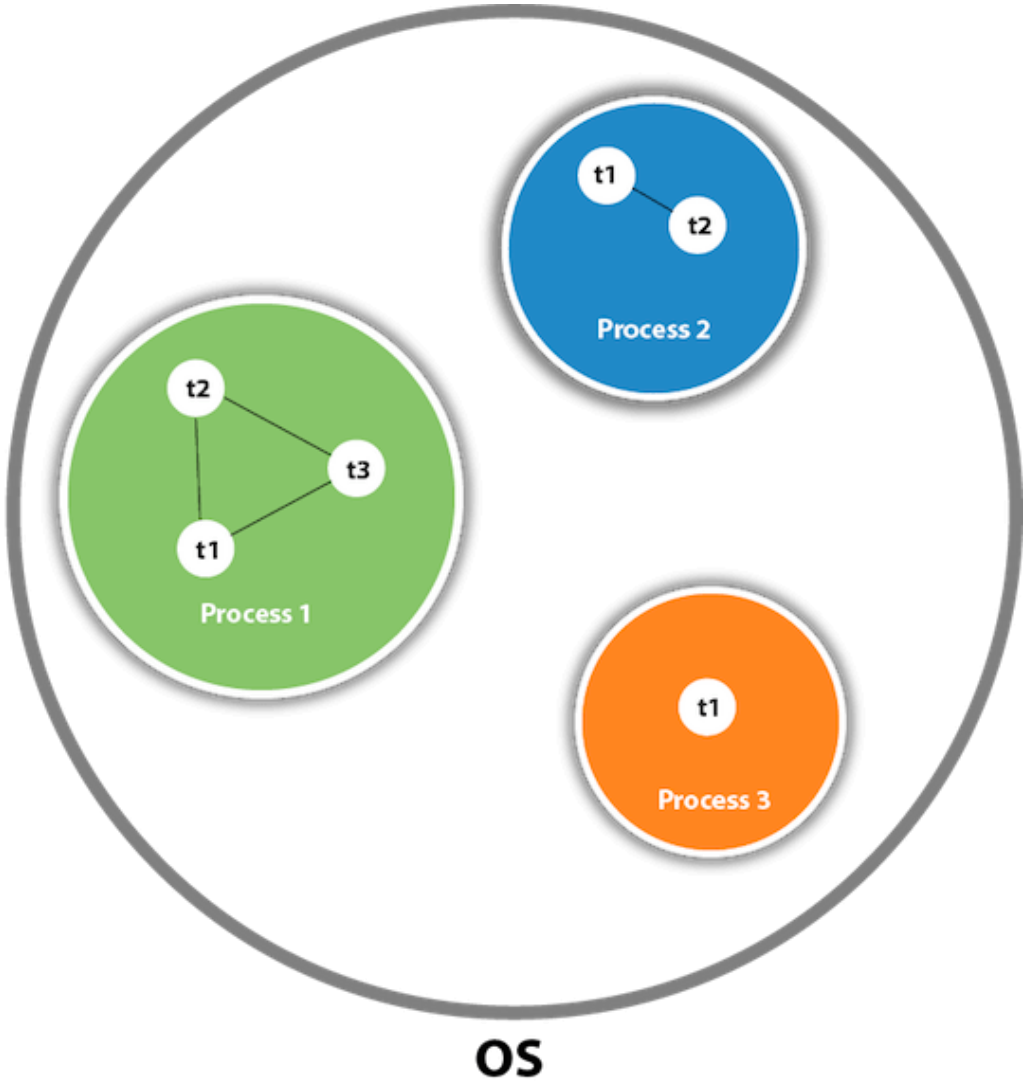
- **Multiple Queues with Different Priorities**
- High-priority queues for interactive tasks (short CPU bursts).
- Low-priority queues for background tasks (long CPU bursts).
- **Priority Adjustment Based on Execution Time**
- Short jobs remain in high-priority queues (fast response).
- Long-running jobs are demoted to lower-priority queues.
- **Aging to Prevent Starvation**
- A long-waiting process in a low-priority queue may be promoted to prevent starvation.

Threads

- A **thread** is the smallest unit of execution within a process. A **process** can have one or multiple threads running in parallel, sharing the same resources.
- A thread is a lightweight process that runs within a larger process.
- Multiple threads within the same process share:

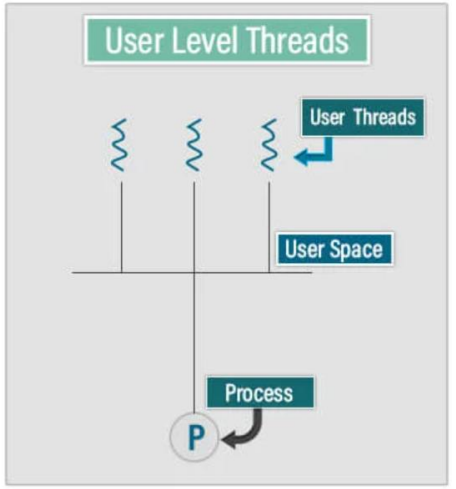
➢The same memory space.

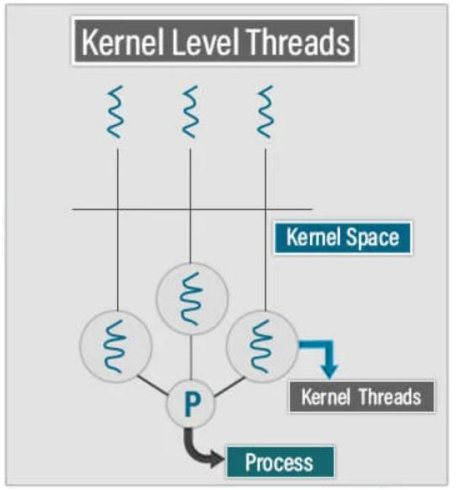
➢The same data.



Threads

- **User-Level Threads (ULT)**
- Managed by user-space libraries, not the OS.
- Faster and more lightweight.
- **Kernel-Level Threads (KLT)**
- Managed by the Operating System.
- More overhead but can take advantage of multiple CPU cores.
- **Hybrid Threads (Two-Level Model)**
- Combination of user-level and kernel-level threads.

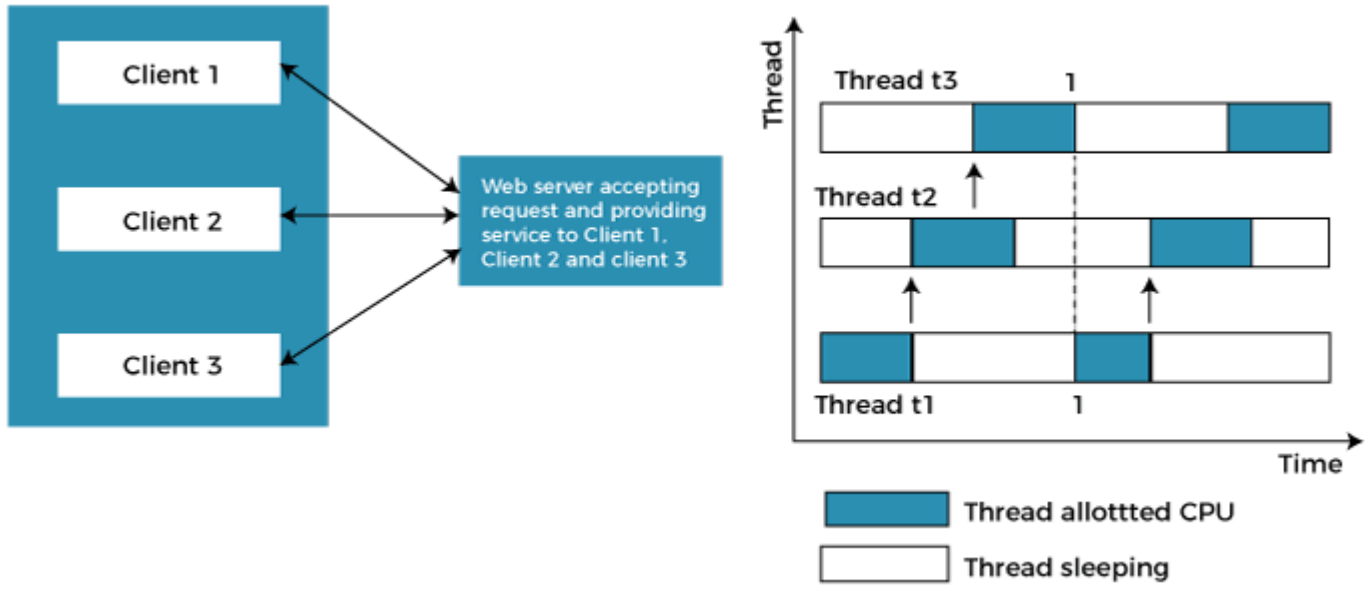


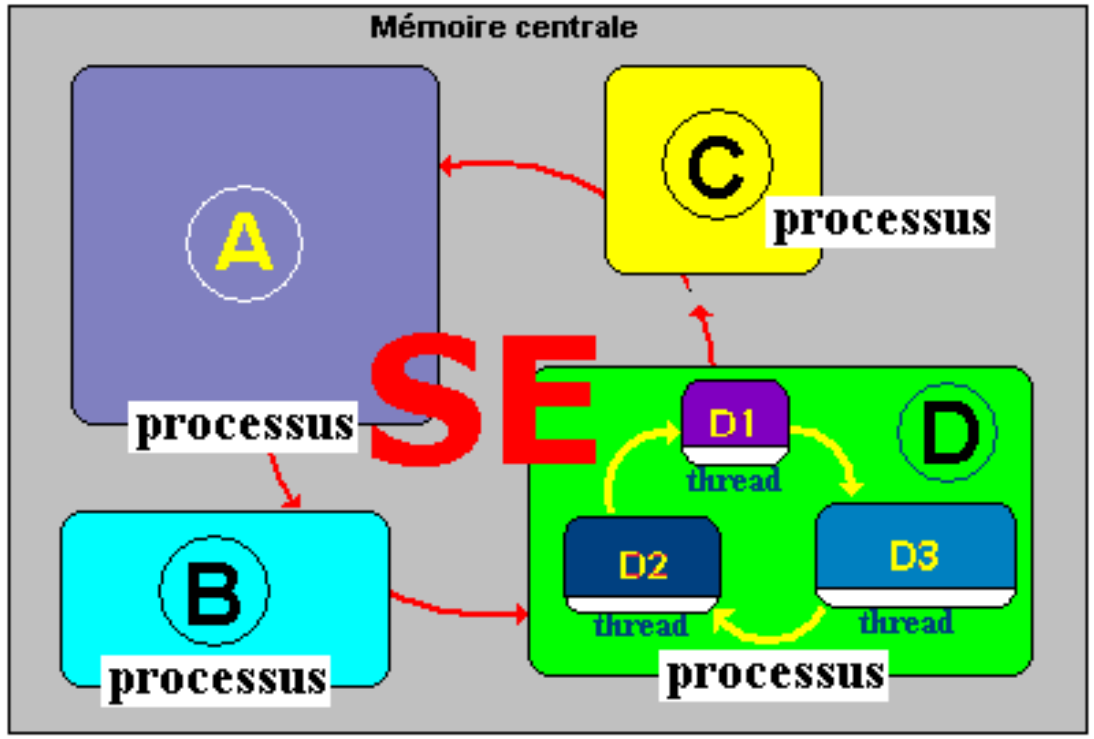


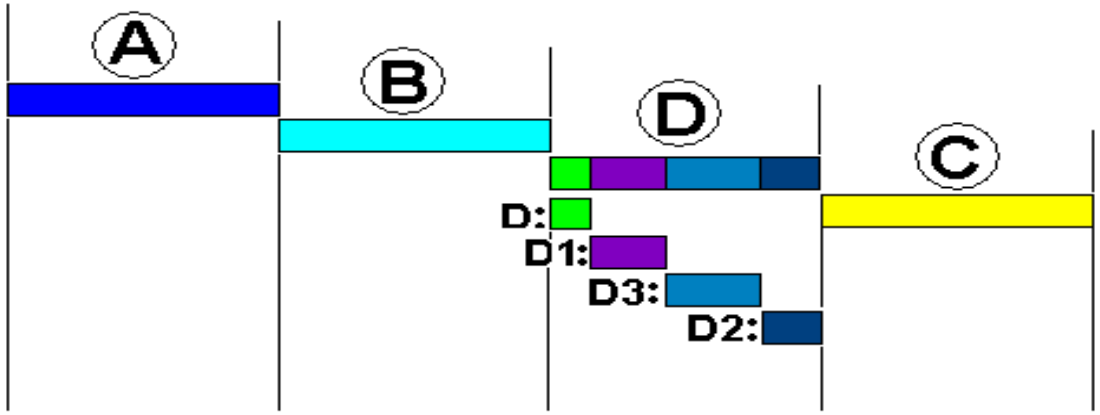
Feature	User-Level Threads (ULT)	Kernel-Level Threads (KLT)
Managed By	User-space libraries (not OS)	Operating System Kernel
Speed	Faster (No system calls)	Slower (Context switch needs system calls)
Parallel Execution	No true parallelism (OS sees only one thread)	True parallelism (Threads run on multiple cores)
Blocking Issue	If one thread blocks, all threads in the process are blocked	If one thread blocks, others continue running
Scheduling	Handled by user-space libraries	Handled by OS scheduler
Context Switching	Faster (No kernel intervention)	Slower (Requires kernel involvement)
Portability	High (Works on different OSes)	Low (Depends on OS support)
Resource Sharing	Threads share process resources	Threads also share, but managed by OS
Examples	POSIX threads (pthreads), Java threads	Windows CreateThread(), Linux clone()

Thread Advantages

- Faster creation & termination compared to full processes.
- Enables concurrent execution of tasks.
- Maximizes performance on multi-core processors (each thread can run on a separate CPU core).







Tranches de temps allouées pendant l'exécution