CHAPTER V: PROCESS AND PROCESSOR MANAGEMENT

I/ Introduction:

The most important of these is the resource that carries out the execution itself, the processor. For this resource, the operating system must provide a set of mechanisms to efficiently and optimally manage the processor's workload and control process execution.

II/ The process:

1/ **Definition** :

A process is a program running in main memory. The process is created at runtime and killed after it has finished.

2/ **what happens after a program launch** :

When the user launches the execution of a program, and when the machine uses the principle of virtual memory and multitasking, then this program goes through the following stages:

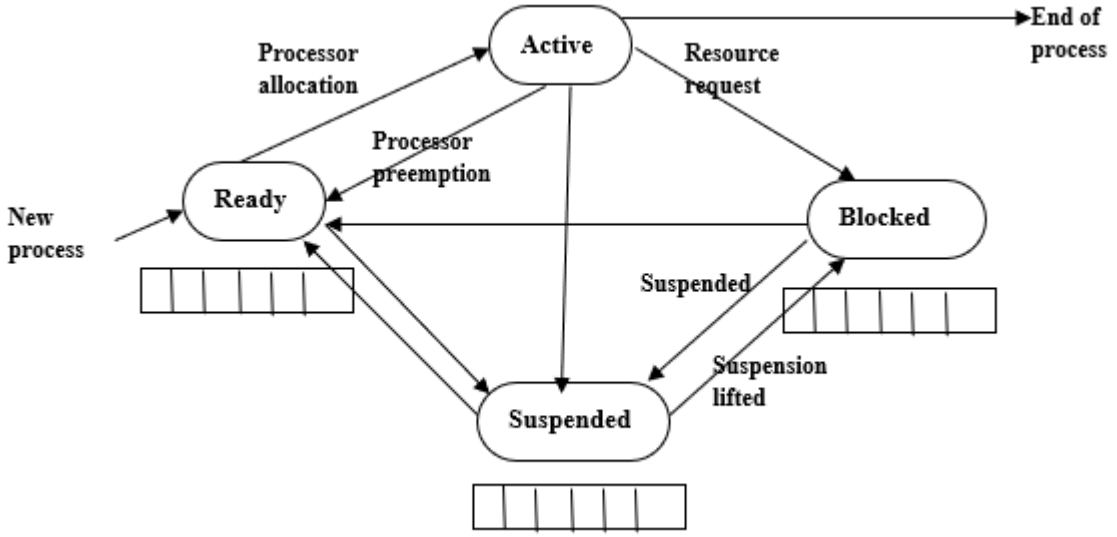
- The system creates a **job** in virtual memory while loading the program.
- Then, according to some criterion or other, this job is admitted into the system and loaded into main memory in its entirety or in part, creating what is known as a **process**.
- The process is then directed to execution in the processor according to specific conditions, where it can change its state several times during execution, until it terminates.

3/ **The change of state of a process**:

During execution, the process can be :

- *Active*: (Elected) i.e. the process has all the necessary resources and is executing at processor level.
- *Ready*: (Eligible) i.e. the process has all necessary resources except the processor process waiting for the processor.
- *Blocked*: the process is waiting for a resource other than the processor, or for an event.
- *Suspended*: the process is temporarily stopped by the user or the OS so that it can no longer compete for the processor.

For each of the ready, blocked and suspended states, a queue is associated with the processes in that state, and is managed by an OS tool.



4/ <u>The process descriptor</u>:

When a process is created (as is a job), its image is created in memory, containing all the information required for the process throughout its execution. This image also represents the process at the level of the various queues and states. This image is called a job descriptor or job control block (JCB) PCB, which contains :

Identity: **External name** | **Internal name**
Priority (dynamic, static)
Context: **CO, PSW, RG, Ac, …**
State
Memory limits: **Page table, occupied areas, addresses, ...**
Resource: **Type, quantity**
Accounting information

The queues used in each state contain the PCBs of the processes, such as :

- The PCB of a new process is placed in the ready queue.
- Wait until selected to enter processor the process becomes active, and a pointer to its PCB in memory is used.
- If the process is waiting for a resource or an event, its PCB enters the blocked process queue.
- If it is suspended, its PCB goes into the suspended process queue.

5/ <u>Process manipulation primitives</u>:

A process is manipulated by necessary primitives that use its PCB, which are system procedures. These primitives are executed indivisibly:

- Process creation.
- Activation of a ready process.
- Suspending a process.

- Destroying a process.

III/ Processor allocation to a process:

Each time the processor becomes inactive, the system must select a process from the ready queue and pass control to it. Two system routines are involved:

1/ The dispatcher:

It is responsible for allocating the processor to a pre-selected process, and for :

- Context switching.
- Branching to the first instruction of the process for execution.

2/ The scheduler:

It selects the process that will have the next execution cycle from the queue of ready processes; it's the process scheduler.

The scheduling operation is performed in the following situations:

- When a new process is created, you need to decide whether to run the parent or child process first.
- A scheduling decision must be made when a process terminates another process must be chosen from the set of ready processes.
- When a process blocks on I/O, semaphore or other (to be seen in 3LMD), another process must be selected for execution.
- When an I/O interrupt occurs, a scheduling decision must also be made.
- If the hardware clock provides periodic interrupts at a frequency of 50 or 60 Hz, for example, a scheduling decision can be made at each clock interrupt (time-sharing system).

There are two levels of scheduling:

- Job scheduling to decide which processes to create.
- Process scheduling to decide which processes are ready to run.

- *Job scheduling*:

It allows you to :

- Monitor job status.
- Allocate necessary resources.
- Recover resources after execution.

- *Process scheduling:*

It allows you to :

- Monitor process status by creating the PCB.
- Decide on processor allocation policy.
- Allocate the processor.
- Take over the processor of a terminated or interrupted process.

3/ Different scheduling policies:

✓ ***Non-preemptive scheduling:***

A non-preemptive scheduling algorithm selects a process, then lets it run until it stalls (either on I/O or waiting for another process) or voluntarily frees up the CPU. Even if it runs for hours, it will not be forcibly suspended. This is because no scheduling decisions are made during clock interrupts. Once the interrupt has been processed, the process that was running before the interrupt is always restarted. Examples of such policies include :

- FCFS policy: first come first served.
- SJF policy: Short Job First the job with the shortest execution time is executed first.
- Priority policy: the process with the highest priority goes first.

✓ ***Preemptive scheduling:***

A preemptive scheduling algorithm selects a process and lets it run for a specified time. If the process is still running at the end of this delay, it is suspended (put in the ready queue), and the scheduler selects another process to run. Pre-emptive scheduling requires an interrupt at the end of the delay to give control of the CPU back to the scheduler. Examples include :

- SRTF policy: Short Remaining Time First the process with the shortest remaining time first.
- Round Robin policy: this policy consists of allocating the processor according to a limited time called the time quantum, where each process enters during this time quantum and then exits and waits for another round of execution. This policy is implemented using a circular queue of ready processes with FIFO management.
- Multi-level queue policy: the ready queue is subdivided into several queues depending on the class of process (batch, interactive, real-time, etc.), each managed independently.