Mohamed Kheider Biskra University
Faculty of Exact Sciences and Natural and Life Sciences
Computer Science department

Course materials

Module: Operating Systems I

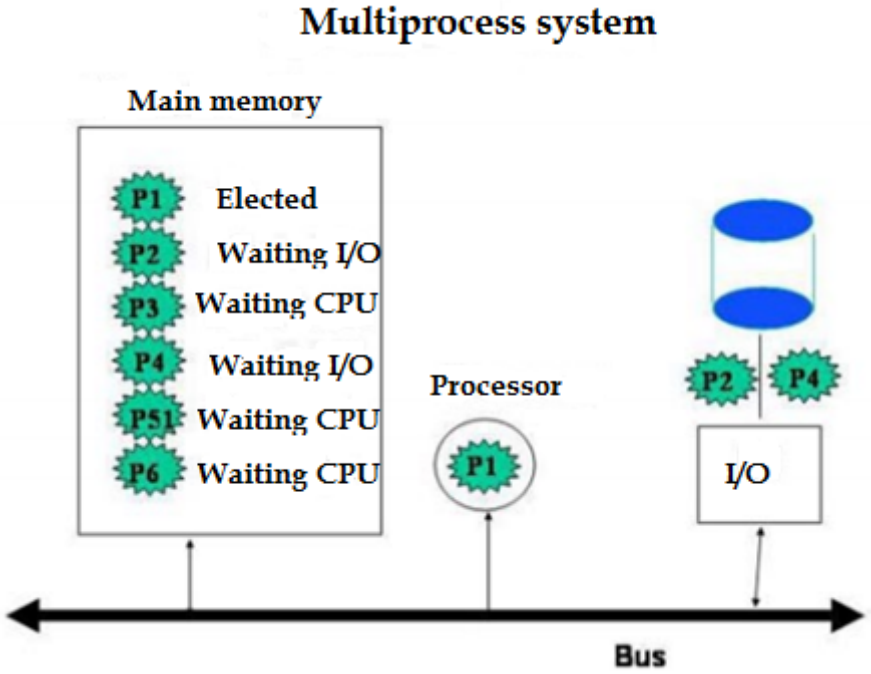
Chapter 3: Process Scheduling

Level: 2LMD
Module leader: Dr. D. Boukhlouf

Academic year 2023/2024

1. Introduction

In a multi programmed system, we "start" several programs at the same time, and when one of them launches an I/O and therefore "waits" for the end of this Input/Output, we take the opportunity to use the CPU to execute another program: we take advantage of the significant speed difference between the execution of the I/O and the CPU. Multiprogramming has become widespread and has become even more complicated with the appearance of interactive "time-sharing" systems. The basic idea behind the notion of "shared time" is to use the difference in speed between the CPU and te "reaction time" of the users: each user is allocated a small "slice" of time of the CPU. If, for example, the CPU can run 10 million instructions per second, with 10 users each will feel like they have a CPU working at 1 million instructions per second. This simplistic diagram can be considerably improved if we take into account the I/O: indeed when a program makes an I/O, it will spend a significant time (on the scale of the CPU) waiting for the end of this I/O: during this waiting time, the CPU can be used to make calculations for other programs, without penalizing the program which made an I/O.



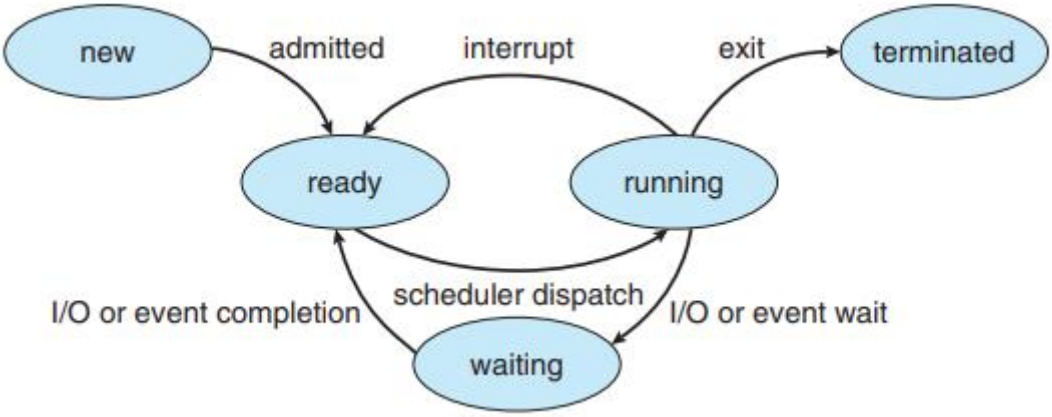
2. Process definition

A process can be defined as a running program. In other words, a program by itself is not a process. A program is a passive entity, like the contents of a file stored on disk, while a process is an active (dynamic) entity, with an instruction counter specifying the next instruction to be executed and a set of associated resources.

Obviously, it is possible to have several different processes associated with the same program. This is the case, for example, of several users who each run a copy of the messaging program. It is also common for one process to in turn spawn multiple processes while running.

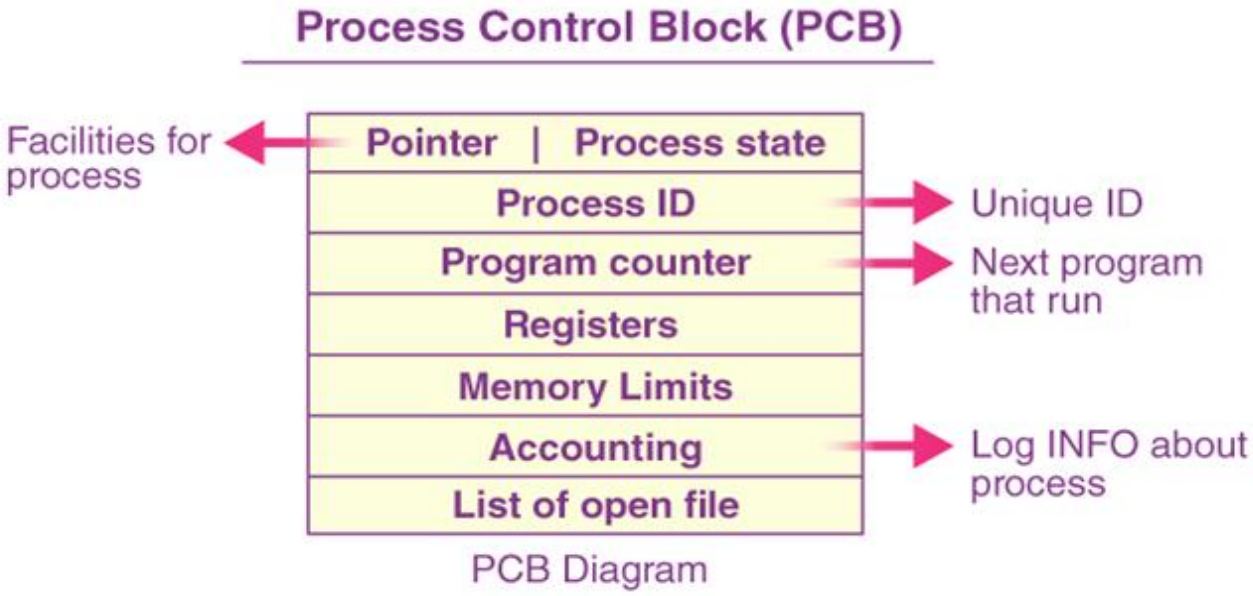
2.1 Process States

During its stay in an OS, a process can change state several times. These states can be summarized in the following diagram:



New (Creation): The process has just been created.
Ready: The process is placed in the ready processes queue, waiting for processor allocation.
Running(Elected): the process has been assigned to a free processor. Its instructions are in execution.
Blocked(Waiting): The process is waiting for an event to occur, such as the completion of an I/O operation or the receipt of a signal.
END: The process has completed its execution.

2.2 Process control block:
Each process is represented in the OS by a PCB (Process Control Block).



It contains several pieces of information about a specific process, such as: The state of the process. Instruction Counter: Indicates the address of the next instruction to be executed by this process. CPU scheduling information: information about process priority. Memory management information: values of base and limit registers, page tables or segment tables. I/O status information: list of I/O devices allocated to this process, a list of open files, etc.

3. Scheduling queues:
To manage the processes during their stay, the OS maintains several queues. We can cite among others:

- Queue of ready processes: This queue contains all the processes waiting for the processor.

- Device Queue: To regulate allocation requests from different devices, one can imagine a queue for each device. When a process requests an I/O operation, it is put in the relevant queue.

4. The scheduler:

The scheduler is a program of the OS which takes care of choosing, according to a given scheduling policy, a process among the processes ready to allocate the processor to it. That is, the scheduler is a module of the operating system which assigns control of the CPU in turn to the various processes in competition according to a policy defined in advance by the designers of the system.

5. Context switching:

Switching the processor to another process requires saving the state of the old process and loading the state saved by the new process. This task is known as context switching.

6. Scheduling criteria:

Different processor scheduling algorithms have different properties and may favor one class of process over another. To choose which algorithm to use in a particular situation, we need to consider the properties of the various algorithms.

Several criteria have been proposed to compare and evaluate the performance of processor scheduling algorithms. The criteria most often used are:

- **CPU usage**: A good scheduling algorithm will be one that keeps the processor as busy as possible.
- **Processing capacity**: It is the quantity of completed processes per unit of time.
- **Waiting time**: This is the time spent waiting in the ready process queue.
- **Response time**: This is the time spent in the queue of ready processes before the first response.
- **Turnaround time**: also known as residence time, is the time taken by a process from its arrival to its termination.
- **Arrival Time:** Time at which the process arrives in the ready queue.
- **Completion Time:** Time at which process completes its execution.
- **Burst Time:** Time required by a process for CPU execution.
- **Turnaround Time (T.A.T):** Time Difference between completion time and arrival time.

  Turn Around Time = Completion Time – Arrival Time
- **Waiting Time (W.T):** Time Difference between turnaround time and burst time.

  Waiting Time = Turnaround Time – Burst Time

7. Scheduling algorithms

The CPU scheduler is used to decide which process (in the ready processes queue) to assign CPU control to. Scheduling strategies can be grouped into two categories: without CPU requisition (non-preemptive strategy) or with CPU requisition (preemptive strategy).

Objectives of Process Scheduling Algorithm:

- Utilization of CPU at maximum level. Keep CPU as busy as possible.
- Allocation of CPU should be fair.
- Throughput should be Maximum. i.e. Number of processes that complete their execution per time unit should be maximized.
- Minimum turnaround time, i.e. time taken by a process to finish execution should be the least.
- There should be a minimum waiting time and the process should not starve in the ready queue.
- Minimum response time. It means that the time when a process produces the first response should be as less as possible.

8. Scheduling modes

Two classes of scheduler:

A. Non-preemptive: (without requisition) Selects a process, then lets it run until it blocks (either on an I/O or waiting for another process) or voluntarily frees the processor. Even if it runs for hours, it won't be forcibly suspended. No scheduling decisions are made during clock interrupts.

B. Preemptive: (with requisition) 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 and the scheduler selects another process to execute.

8.1 Scheduling without CPU requisition (without preemption)

In this category, a process retains control of the CPU until it crashes or terminates. This approach corresponds to the needs of batch jobs (batch systems). There are several algorithms in this category:

8.1.1 First Come First Served (FCFS: First come First Served) (also called FIFO)

The simplest processor scheduling algorithm is the First Come First Served (FCFS) algorithm. With this algorithm, the processor is allocated to the first process that requests it. FCFS policy implementation is easily managed with a FIFO queue. When a process enters the ready process queue, its PCB is chained to the tail of the queue. When the processor becomes free, it is allocated to the processor at the top of the queue.

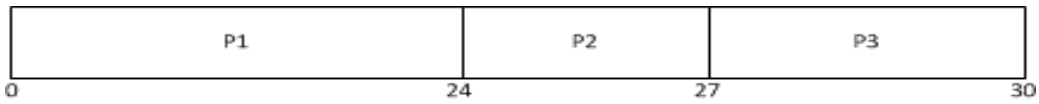
Example :

•We consider all the following processes:

Process	CPU Burst
P1	24
P2	3
P3	3

It is assumed that the processes arrive at time 0 and admitted in the order: P1 , P2 , P3

•The Gantt chart for the FCFS scheduling of this set is:



•Waiting time for P1 = 0; P2=24; P3 = 27

•Average waiting time: (0 + 24 + 27)/3 = 17

If the processes had arrived in the order P2, P3 and P1, the results would be different:

The average waiting time would be: (0+3+6)/3=3 units.

Thus the average waiting time with an FCFS policy is generally not minimal and can vary substantially if the execution times of the processes vary a lot.

Critique of the method:

The FCFS method tends to penalize short jobs: The FCFS algorithm does not perform requisitions. That is to say that once the processor has been allocated to a process, this one keeps it until it releases it, either by terminating, or after having requested an I/O.

The FCFS algorithm is particularly inconvenient for time-sharing systems, where it is important for the user to obtain the processor at regular intervals. It may seem disastrous to allow a process to hold onto the processor for an extended period of time.

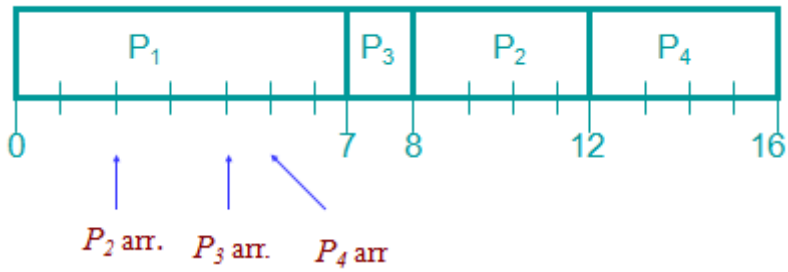
8.1.2 Shortest first: Shortest Job First (SJF)

This algorithm (in English Shortest Job First: SJF) assigns the processor to the process with the shortest execution time. If several processes have the same duration, a FIFO policy will then be used to decide between them.

Example :

Process	Arrival Time	CPU Burst
P1	0	7
P2	2	4
P3	4	1
P4	5	4

The Gantt chart:



Average waiting time = (0+(8-2)+(7-4)+(12-5))/4
(0 + 6 + 3 + 7)/4 = 4

Critique of the method:

The SJF algorithm has been proven to be time-optimal in the sense that it achieves the shortest waiting time for a given set of processes. However, this algorithm is difficult to implement for a simple reason: How can we know the execution time of a process in advance?
Solution: Prediction technique.

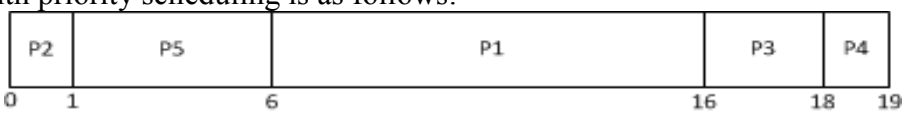
8.1.3 Scheduling with priority:

This algorithm associates each process with a priority, and the processor will be assigned to the process with the highest priority. This priority varies between systems and can range from 0 to 127.
Priorities can be defined according to several parameters: the type of process, time limits, memory limits, etc.

Example: We have 5 processes with different priorities, as shown in this table:

Process	CPU Burst	Priority
P1	10	3
P2	1	1
P3	2	4
P4	1	5
P5	5	2

•The Gantt chart with priority scheduling is as follows:



The average waiting time is = (0+1+6+16+18)/5=8.2 time units.

Critique of the method:

A non-preemptive priority scheduling algorithm places the new process at the head of the ready process queue:

•A scheduling algorithm with priority poses a major problem

–Infinite blocking or starvation (starvation)

–Can leave low priority processes waiting indefinitely

•Solution:

– Aging: A technique of gradually increasing the priority of processes that have been waiting in the system for a long time.

8.2 Scheduling with CPU requisition (with preemption)

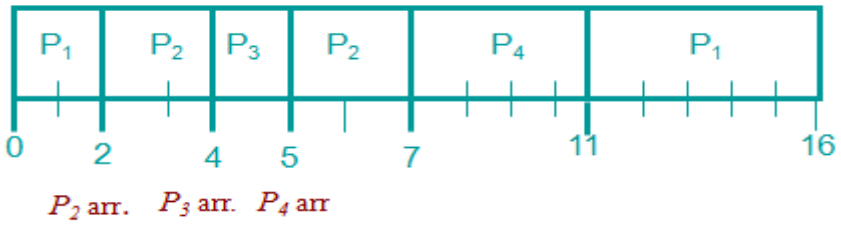
In this category, the scheduler can remove the CPU from a process before the latter hangs or terminates in order to allocate it to another process. At each quantum (unit of time) the scheduler chooses from the list of ready processes a process to which the CPU will be allocated. This approach corresponds to interactive systems.

8.2.1 The shortest remaining time (SRTF: Shortest Remaining Time First)

It is the preemptive version of the SJF algorithm. At the start of each quantum, the CPU is allocated to the process that has the smallest remaining execution time.

Example of SJF with preemption (SRTF)

Process	Arrival Time	CPU Burst
P1	0	7
P2	2	4
P3	4	1
P4	5	4

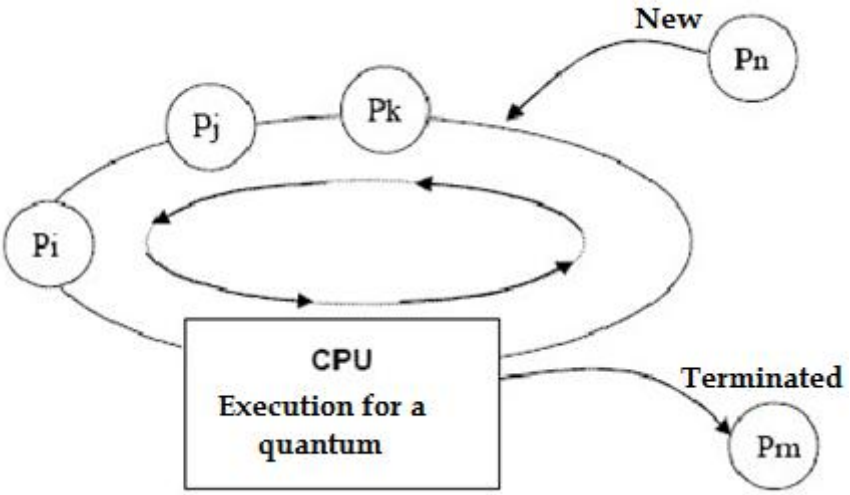


Average waiting time = (9 + 1 + 0 +2)/4 = 3

P1 waits from 2 to 11, P2 from 4 to 5, P4 from 5 to 7

8.2.2 Round Robin Algorithm:

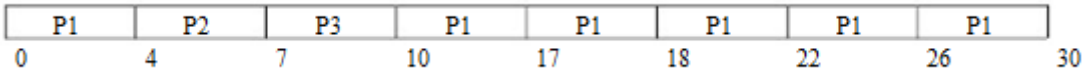
CPU control is assigned to each process for a slice of time Q, called a quantum, in turn. When a process runs during a quantum, the context of the latter is saved and the CPU is allocated to the next process in the list of ready processes (see the figure below). In practice, the quantum ranges between 10 and 100 ms.



Example: We have 3 processes P1, P2 and P3 having execution times of 24, 3 and 3 ms respectively.

Process	CPU Cycle (ms)
P1	24
P2	3
P3	3

Using a Round Robin algorithm, with a quantum of 4 ms, we obtain the following Gantt chart:

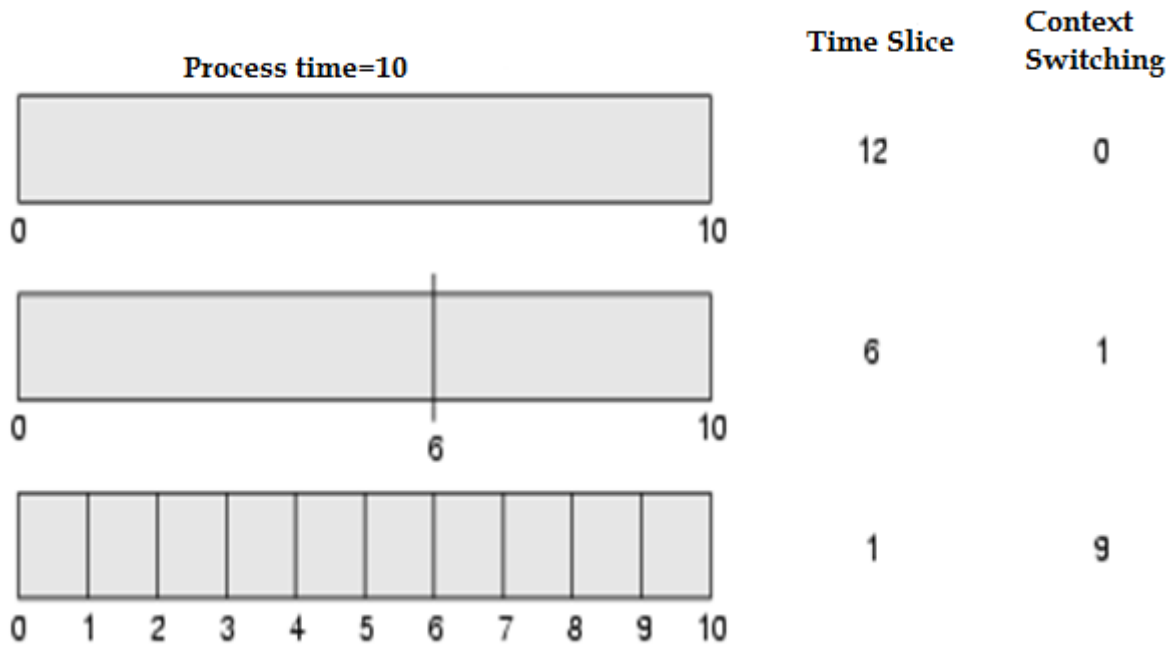


The average waiting time is: (6+4+7)/3 = 17/3 = 5.66 ms

Critique of the method

The performance of this policy depends on the value of the quantum. A too large quantum increases the response time, the Round Robin policy would be similar to that of the FCFS. A quantum that is too small multiplies the switchings of context, which, from a certain frequency, can no longer be neglected.

Example: We have a process P whose execution time is 10 ms. Let's calculate the number of context switches needed for a quantum equal respectively to: 12, 6 and 1.



8.2.3 Priority with preemption

Example:

It is assumed in this example th18at the smallest value corresponds to the highest priority.

PID	Arrival time	CPU Burst	Priority
A	0	10	3
B	0	1	1
C	10	2	2
D	0	1	4
E	0	5	2

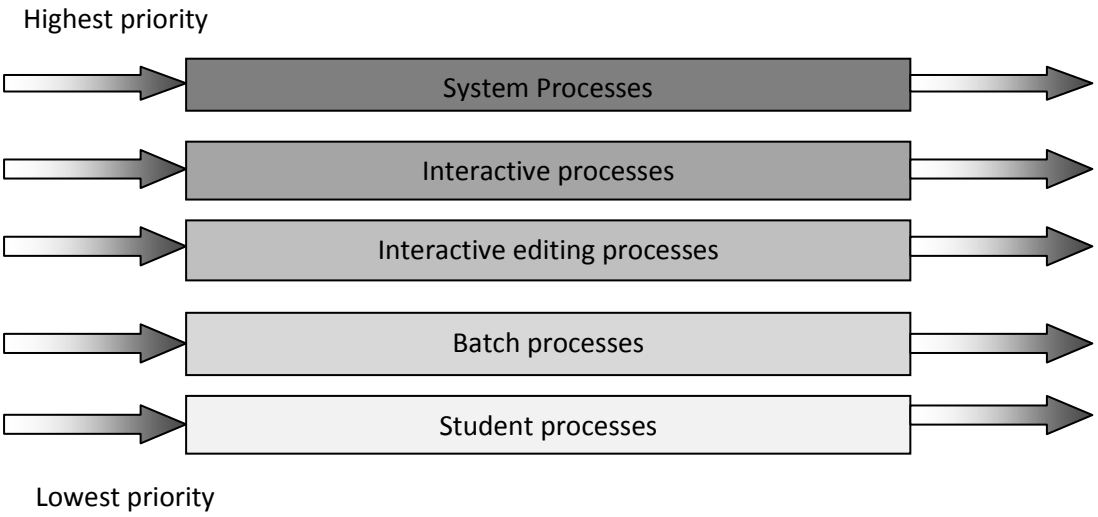
B	E	A	C	A	D
0 1	6	C↑10	12	18	19

Average waiting time= (8+0+0+18+1)/5=5,4

9. Scheduling with multilevel queues:

Another class of scheduling algorithms has been developed for situations where one can easily classify processes into different groups. For example, it would be interesting to make a distinction between foreground (interactive) processes and background (batch) processes. Indeed, these two types of processes have different needs in terms of response time and therefore may need to be scheduled differently. In addition, foreground processes can take priority over background processes.

Thus, a scheduling algorithm with multilevel queues splits the queue of ready processes into several separate queues. The following figure gives an example of the breakdown of these queues (e.g.: main server of a university):



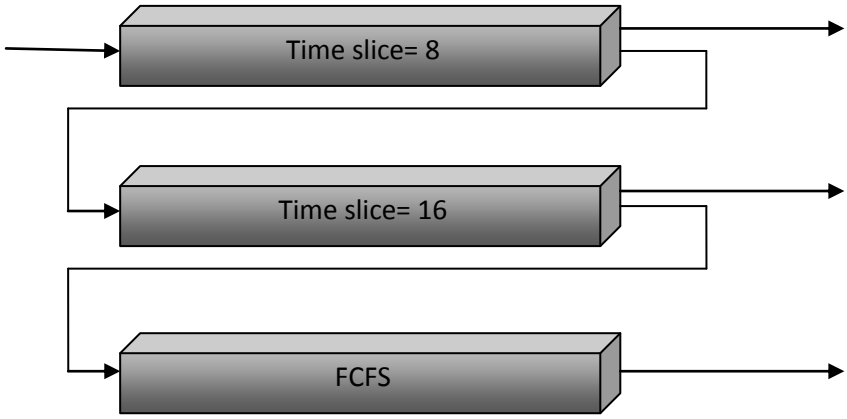
Processes are permanently assigned to a queue usually based on certain process properties, such as: process type, memory size, priority, etc. Each queue has its own scheduling algorithm. For example, the queue of system processes is scheduled according to the highest priority algorithm, those of interactive processes are managed according to the Round Robin algorithm and the batch process queue is managed according to the FCFS algorithm. On the other hand, there must be scheduling between the queues themselves.
Consider, for example, the following set of multilevel queues:

1. Systems processes
2. Interactive processes
3. Batch process
4. User processes

Each queue has absolute priority over lower level queues. For example, no process in the batch process queue will be able to run unless the system and interactive process queues are all empty. Also, if an interactive process enters the system while a batch process is running, the batch process must be terminated.
Another way to do this would be to assign time slices to queues. Each queue obtains a certain part of the processor time, which must be scheduled between the different processes that compose it. For example, you can allocate 80% of the processor time to the foreground process queue and 20% to the background process queue.

10. Scheduling with multilevel queues and feedback:

Normally, in an algorithm with multilevel queues, processes are permanently assigned to a queue as soon as they enter the system. Processes do not move between queues. This organization has the advantage of a low overhead due to scheduling, but it lacks flexibility. Thus, scheduling with multilevel feedback queues allows processes to move between queues. The idea comes down to separating the processes according to the evolution of their characteristics in the system.
Example: A system has 3 multilevel queues: Queue 0, Queue 1 and Queue 2. Queue 0 has the highest priority. Queues 0 and 1 are managed according to the Round Robin policy. Queue 2 is managed using the FCFS technique.



A process entering the system will be placed in queue 0. The process is given a time slice of 8 ms. If it does not finish, it is moved to queue 1. If queue 0 is empty, a 16 ms time slice is given to the header process of queue 1. If it does not finish, it is interrupted and put in queue 2. Processes in queue 2 are executed only when queues 0 and 1 are empty.

References

SILBERSCHATZ, A. and PB GALVIN, Operating System Concepts. 8th Edition, Addison Wesley.2012.