University of Tlemcen

Computer Science Department

1st Year Engineer

"Introduction to the operating systems 2"

Sessions 5, 6 and 7: CPU scheduling

Academic year: 2023-2024

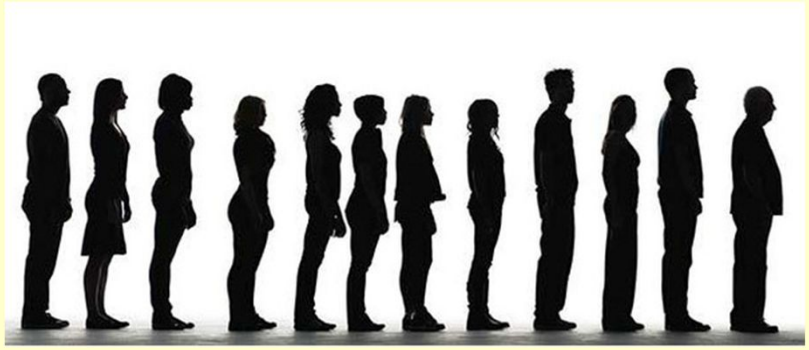
Outline

- Process scheduling techniques :
  - *Criteria (fairness, efficiency, response time, execution time, throughput)*
- Scheduling algorithms (some of the most commonly used):
  - *First Come First Served (FCFS) algorithm.*
  - *Shortest Job First (SJF) algorithm.*
  - *Shortest Remaining Time (SRT) algorithm.*
  - *Round Robin RR.*
  - *Algorithm with priority.*

Motivations

- When a computer is multi-programmed, it frequently has several processes/threads competing for processor time.

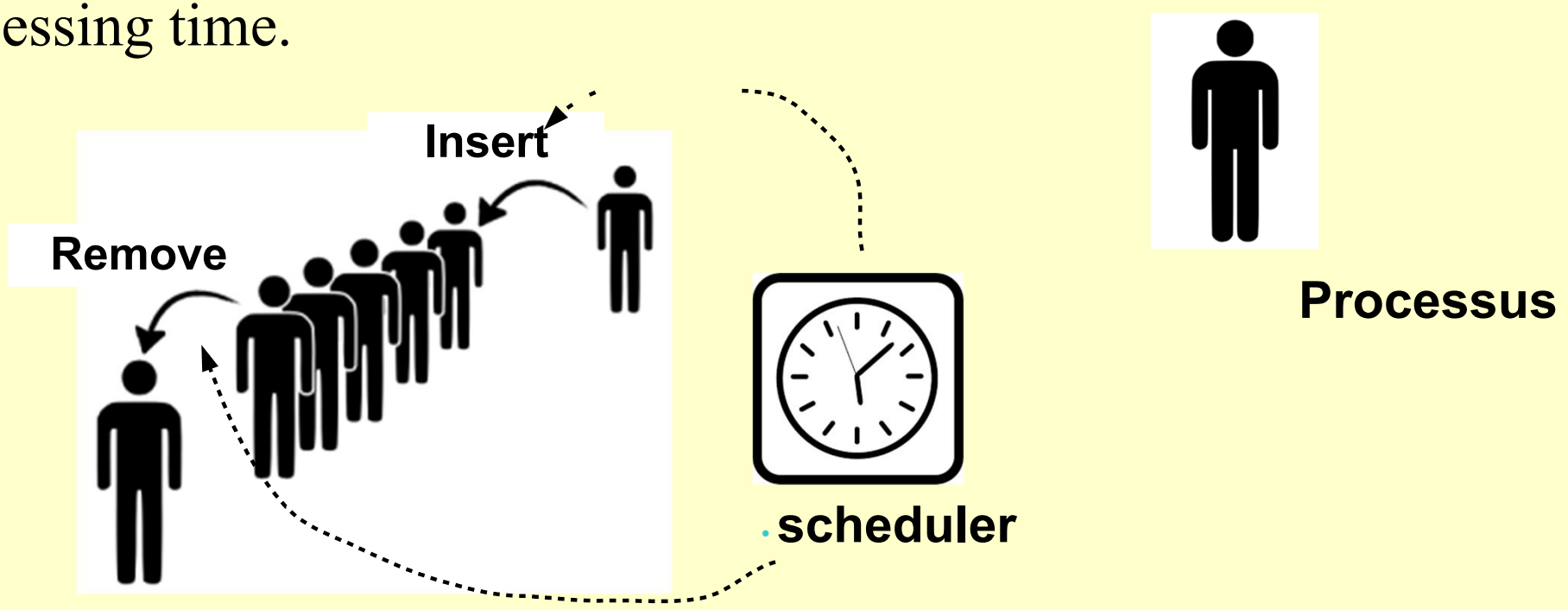
→ If there is only one processor, a choice must be made as to the next process to run



→

Definition

- The part of the operating system that makes this choice is called the ***Scheduler*** and the algorithm it uses is called the ***Scheduling algorithm***.
- As well as selecting the appropriate process to run, the scheduler must also ensure that it makes efficient use of the processor, as switching from one process to another is costly in terms of processing time.



Challenges

Response time

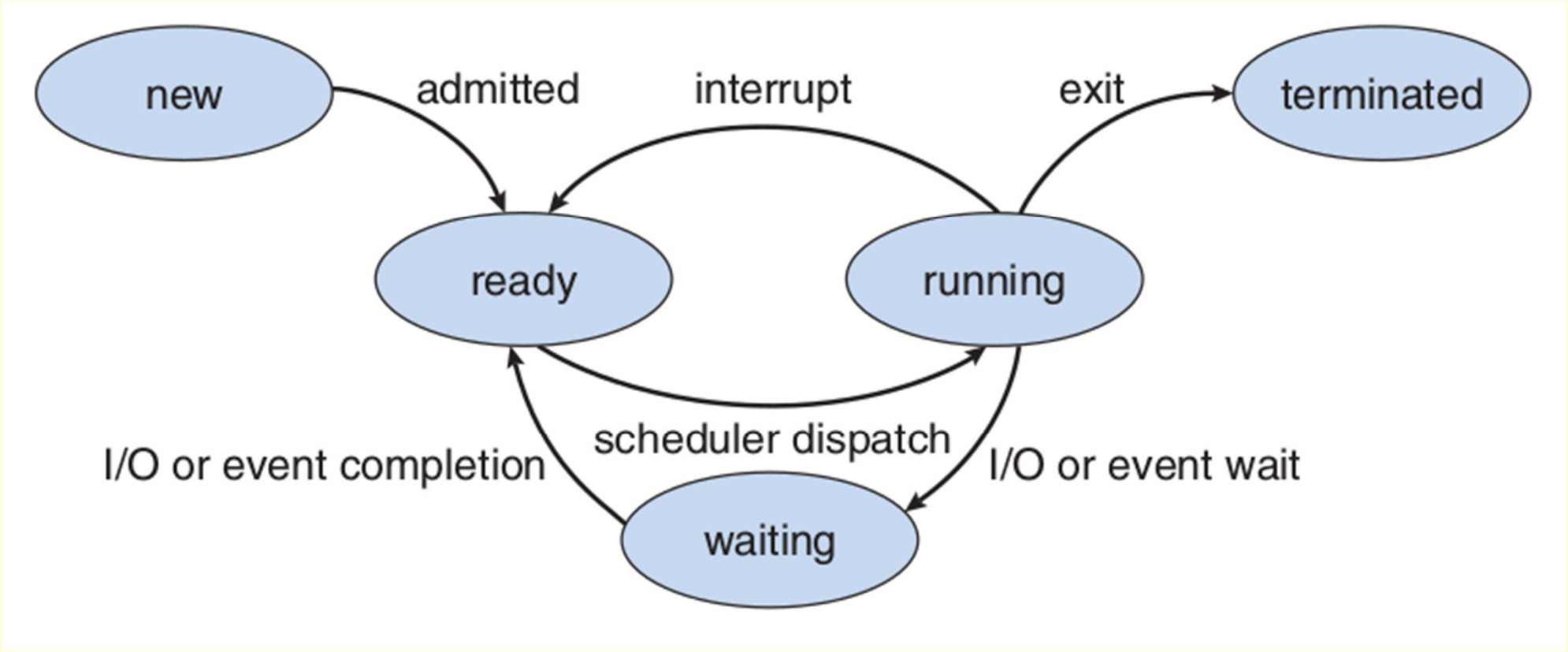
Time to change context

<u>Efficiency: Making the best use of the processor</u>

Objectives of the scheduler

- The objectives of a scheduler for a multi-user system include :
  - ***Fairness**: Ensure that each process waiting to run receives its share of processor time.*
  - ***Performance**:*
    - *Ensure that the processor is always busy,*
    - *Ensuring that all parts of the system are busy,*
    - *Minimize response time.*
  - *Take priorities into account.*
  - ***Reactive**: responding quickly to requests (example: commands).*
  - ***Throughput**: number of processes per unit of time.*

Process states and scheduling
When should you schedule?



- **When a new process is created**

  → a decision must be made as to whether to run the parent process or the child process first.

- **When a process terminates**

  → another process must be chosen from the ready processes.

- **When a process stalls**

  → another process must be selected to run.

- **When an I/O interrupt occurs**

  → a scheduling decision must be made among the processes that were blocked waiting for I/O.

Interruption ?

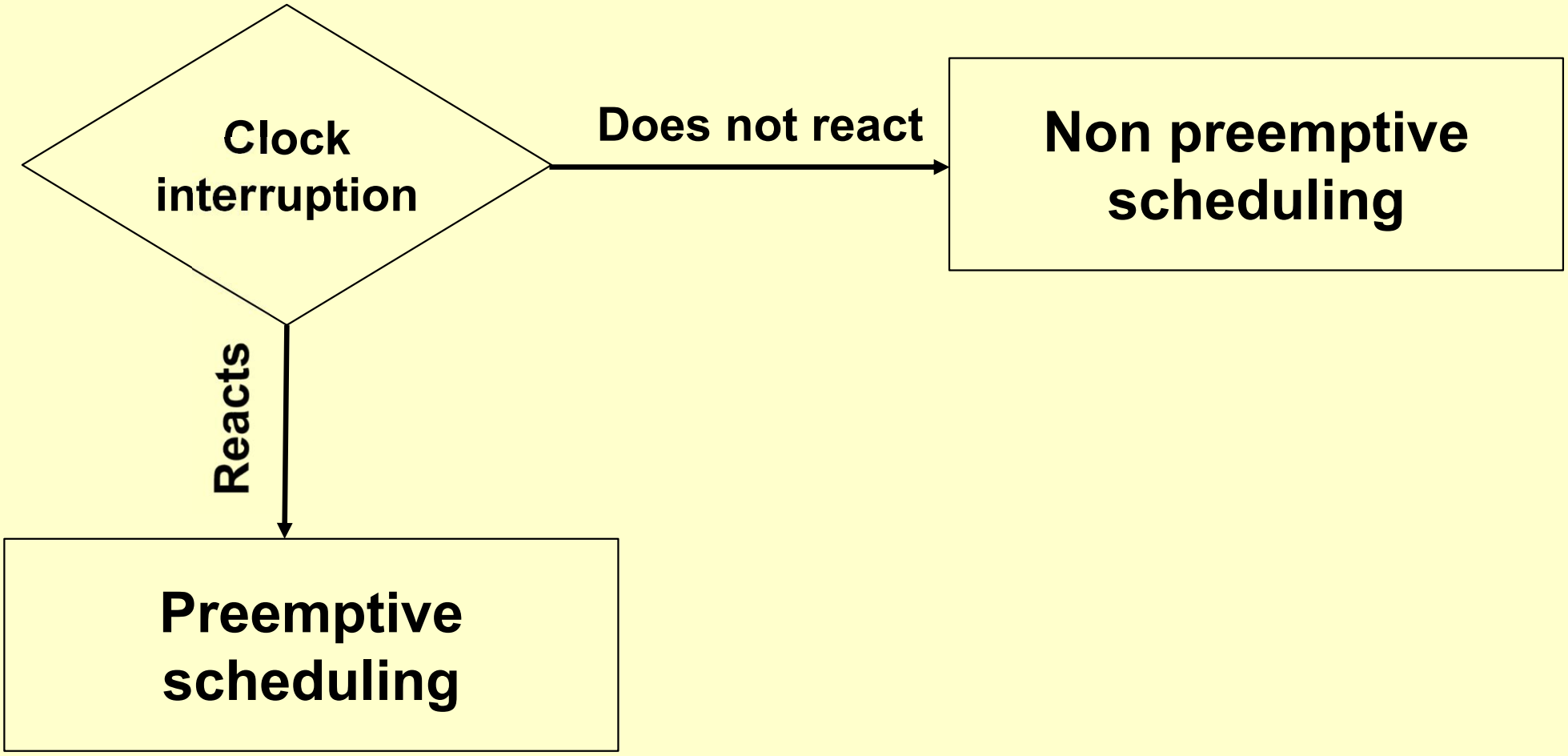
- Interruption
  - *Temporary cessation of execution.*
- Category: Hardware vs. Software
- Software interrupt :
  - *If the CPU detects an error in the processing of an instruction (division by zero for example).*
  - *Stops execution to run a specific ISR (Interrupt Service Routine) for each type of error encountered (memory overflow, division by zero, etc.).*

Hardware Interruption
IRQ: Interrupt Request

- Hardware interruptions:
  - *Generated by peripherals (keyboard, disk, USB, etc.),*
  - *Can be masked (forbidden or authorized).*
- Interrupts: the peripheral signals events to the CPU by interruption.
- The CPU momentarily stops execution of a process to execute the peripheral's request.
  - *Triggers scheduling between blocked processes while waiting for the I/O in question.*

Interruption and Scheduling

- Clock interrupts are used to switch tasks in multitasking systems.
- Typically, a periodic interrupt is triggered by a clock (often 100 or 1,000 Hz), and the scheduler is then put into action.


Clock interruption
Does not react
Non preemptive scheduling
Reacts
Preemptive scheduling

Non-preemptive scheduling algorithms

- Selects a process, then lets it run until it hangs, or voluntarily frees up the processor.
  - *Even if it runs for hours, it will not be forcibly suspended.*
  - *No scheduling decisions are made during clock interruptions.*

Preemptive scheduling algorithms

- Selects a process and lets it run for a specified time.
  - *If the process is still running at the end of this time, it will be suspended.*
  - *The scheduler selects another process to run.*

Non-preemptive schedulers

- In a non-preemptive or non-requisition scheduling system, the operating system chooses the next process to run:
  - *First-Come-First-Served (FCFS),*
  - *or Shortest Job First (SJF).*
- It allocates the processor to it until it completes or is blocked (waiting for an event).

  → There is no requisition.

First-Come, First-Served (FCFS)

- This scheduling algorithm is implemented via a FIFO queue
  - → *very easy code to implement*
  - *On the other hand, this is a strategy that can generate large and highly variable average waiting times.*
- Example:
  - *Consider processes P_1, P_2, P_3 with the following execution times. These processes all arrive at time 0.*

P_1	24
P_2	3
P_3	3

- If the processes arrive in the order P1, P2, P3, we get the following Gantt diagram:

- The waiting time for P1 is 0 ms, for P2 it is 24 ms and for P3 it is 27 ms.
- The average waiting time is :

$$Avg\_Wait\_Time = \frac{\sum_{i=1}^{n} Wait\_Time_i}{n} = \frac{0+24+27}{3} = 17\ ms$$

- Assuming that the processes arrive in the order P_2, P_3, P_1, we get the following Gantt diagram:

- The average waiting time becomes :

$$Avg\_Wait\_Time = \frac{\sum_{i=1}^{n} Wait\_Time_i}{n} = \frac{6 + 0 + 3}{3} = 3\ ms$$

Shortest Job First (SJF)

- The scheduler chooses the shortest process (shortest execution time) from the batch of processes to be executed.
- Strategy offers the minimum average waiting time.

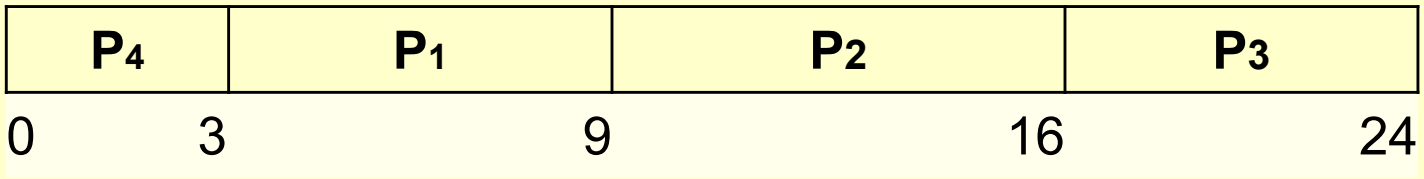
Process Scheduling: Cooperative O.S

- For example, processes P_1, P_2, P_3 and P_4 all arrive at time 0 and have the following execution times:

P_1	6
P_2	7
P_3	8
P_4	3

Process Scheduling : Scheduling Queues

- With SJF, we get the following Gantt diagram:



- P_1 waits 3 ms, P_2 9 ms, P_3 16 ms and P_4 0ms.
- The average waiting time is :

$$Avg\_Wait\_Time = \frac{\sum_{i=1}^{n} Wait\_Time_i}{n} = \frac{3 + 16 + 9 + 0}{4} = 7\ ms$$

Non-preemptive schedulers: Metrics

- Metrics :
  - *dwell time (Response time) = termination time - entry time*
  - *Wait time = dwell time - execution time*

Non-preemptive schedulers: Exercise I

- Consider five processes A, B, C, D and E, whose respective execution times and arrival times are as follows:

Process	Execution time	Arrival time
A	3	0
B	6	1
C	4	4
D	2	6
E	1	7

- Draw a diagram illustrating the execution (Gantt diagram) and calculate the response time of each process, the average response time, the waiting time and the average waiting time according to the following scheduling policies:
  - *First Come First Served (FCFS)*
  - *Shortest Job First (SJF)*

Correction: FCFS strategy

A	A	A	B	B	B	B	B	B	C	C	C	C	D	D	E
1				5					10					15	

- *At time 0, only process A is in the system and is executing.*
- *At time 1, process B arrives but has to wait for process A to finish executing, as it still has 2 units of time left.*
- *B then runs for 6 units of time.*
- *At times 4, 6 and 7 processes C, D and E arrive, but B has not yet finished executing.*
- *Once B has finished, C, D and E enter the system in order.*

- Dwell time calculation for each process::

$$\boldsymbol{Stay\ time = Termination\ time\ - Arrival\ time}$$

Process	***Termination time***	***Arrival time***	***Stay time***
A	3	0	3
B	9	1	8
C	13	4	9
D	15	6	9
E	16	7	9

- The average time of stay is:

$$\frac{\sum_{i=1}^{n} Stay\_time_i}{n} = \frac{3+8+9+9+9}{5} = 7.6\ ms$$

Cont'

- **The waiting time of each process:**

$$Waiting\ time = Stay\ time\ - Execution\ time$$

Process	Stay time	Execution time	Waiting time
A	3	3	0
B	8	6	2
C	9	4	5
D	9	2	7
E	9	1	8

- **The average waiting time :**

$$= \frac{\sum_{i=1}^{n} Wait\_Time_i}{n} = \frac{0+2+5+7+8}{5} = 4.4\ ms$$

- There are five processes running in 16 time units, so 16/5=3.2 time units per process on average.

Correction : SJF strategy

- The shortest first among the arriving processes, the execution scheme is as follows:

A	A	A	B	B	B	B	B	B	E	D	D	C	C	C	C
1				5					10					15	

Correction : SJF strategy

- The stay time of each process:

$$\boldsymbol{Stay\ time = Termination\ time\ - Arrival\ time}$$

Process	*Termination time*	*Arrival time*	*Stay time*
A	3	0	3
B	9	1	8
E	10	7	3
D	12	6	6
C	16	4	12

- Le temps moyen de séjour est:

$$= \frac{\sum_{i=1}^{n} tps\_se_i}{n} = \frac{3+8+3+6+12}{5} = 6.4\ ms$$

Cont'

- **The waiting time of each process:**

$$Waitin\ time = Stay\ time\ - Execution\ time$$

Process	Stay time	Execution time	Waiting time
A	3	3	0
B	8	6	2
E	3	1	2
D	6	2	4
C	12	4	8

- **The average waiting time is :**

$$= \frac{\sum_{i=1}^{n} Wait\_Time_i}{n} = \frac{0+2+2+4+8}{5} = 3.2\ ms$$

Cont'

- There are five processes running in 16 units of time, so 16/5=3.2 units of time per process.

Preemptive schedulers

- A preemptive scheduler is a scheduler that acts with requisition.
- To ensure that no process runs for too long

  → an electronic clock periodically generates an interruption.
- At each clock interruption, the operating system takes control again and decides :
  - *whether the current process should continue to run,*
  - *or whether it should be suspended to make way for another process.*

- If the current process has to suspend execution in favor of another :
  - *The OS must first save the process context,*
  - *before loading the context of the process to be launched.*
- This is known as context switching.
  - *This backup is necessary to be able to continue execution of the suspended process at a later date.*

- The time taken to allocate the processor to the process is called the ***quantum***.
- Switching between processes must be fast, i.e. require a time well below the quantum.

Scheduling: Shortest Next Remaining Time (SNRT)

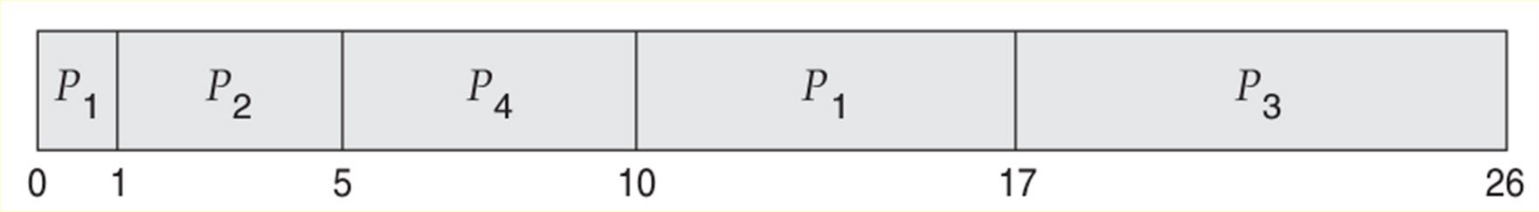
- SRT is the preemptive version of the SJF algorithm.
  - *If a process whose execution time is shorter than the rest of the execution time of the process currently being processed, then it will take its place.*

- Let's consider four processes P_1, P_2, P_3, and P_4 :

Process	Arrival time	Execution time
P_1	0	8
P_2	1	4
P_3	2	9
P_4	3	5

- P_1 starts at t=0, it is the only process in the queue P_2 arrives at t=1ms.
- The time remaining to P_1 is 7ms > time requested by P_2, 4ms → P_2 is executed and P_1 returns to the queue.

- The execution of the various processes is based on the following Gantt chart:



- Calculating stay and waiting times :

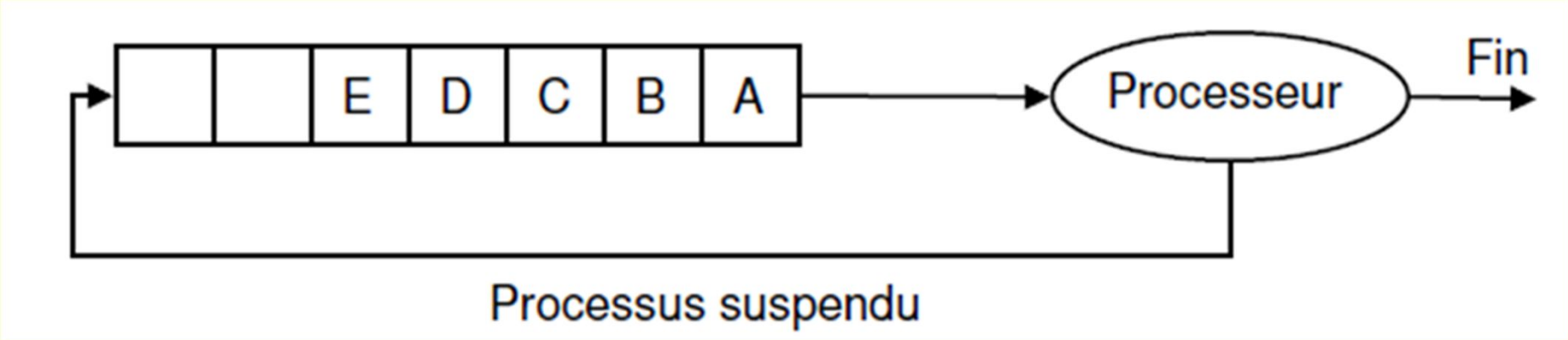
Process	Arrival time	Execution time	Stay time	Waiting time
P1	0	8	17	9
P2	1	4	4	0
P3	2	9	24	15
P4	3	5	7	2

- The average waiting time is :

$$= \frac{\sum_{i=1}^{n} Wait\_Time_i}{n} = \frac{9 + 0 + 15 + 2}{4} = 6.5\, ms$$

Circular scheduling : Round Robin (RR)

- The round robin algorithm is a simple, reliable and widely used algorithm.
- It stores the list of ready processes in a FIFO-type queue.



Selecting the process to run in RR

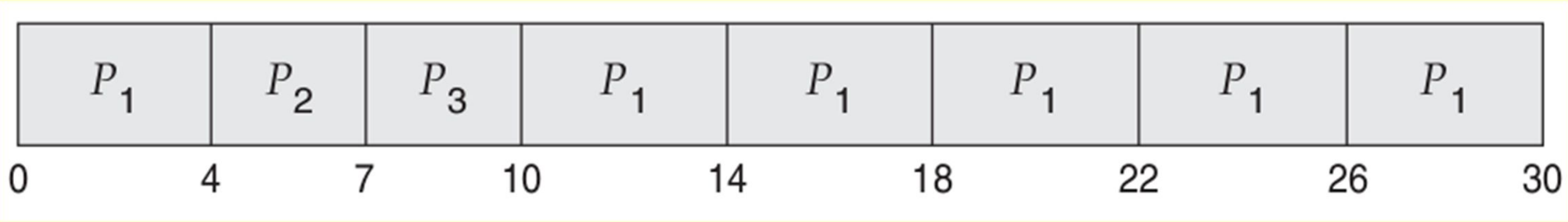
- It allocates the processor to the process at the head of the queue, for a quantum of time.
- If the process stalls or terminates before the end of its quantum, the processor is immediately allocated to another process (the one at the head of the queue).
- If the process does not terminate at the end of its quantum, its execution is suspended.
  - *The processor is allocated to another process (the one at the head of the queue).*
  - *The suspended process is inserted into the queue.*
  - *Processes that arrive or move from the blocked state to the ready state are inserted into the queue.*
  - *To avoid starvation, a new process is inserted at the end of the queue, so as not to duplicate existing processes.*

Example : Round Robin

- Let's consider the processes P_1, P_2 and P_3 :

Process	Execution time (ms)
P_1	24
P_2	3
P_3	3

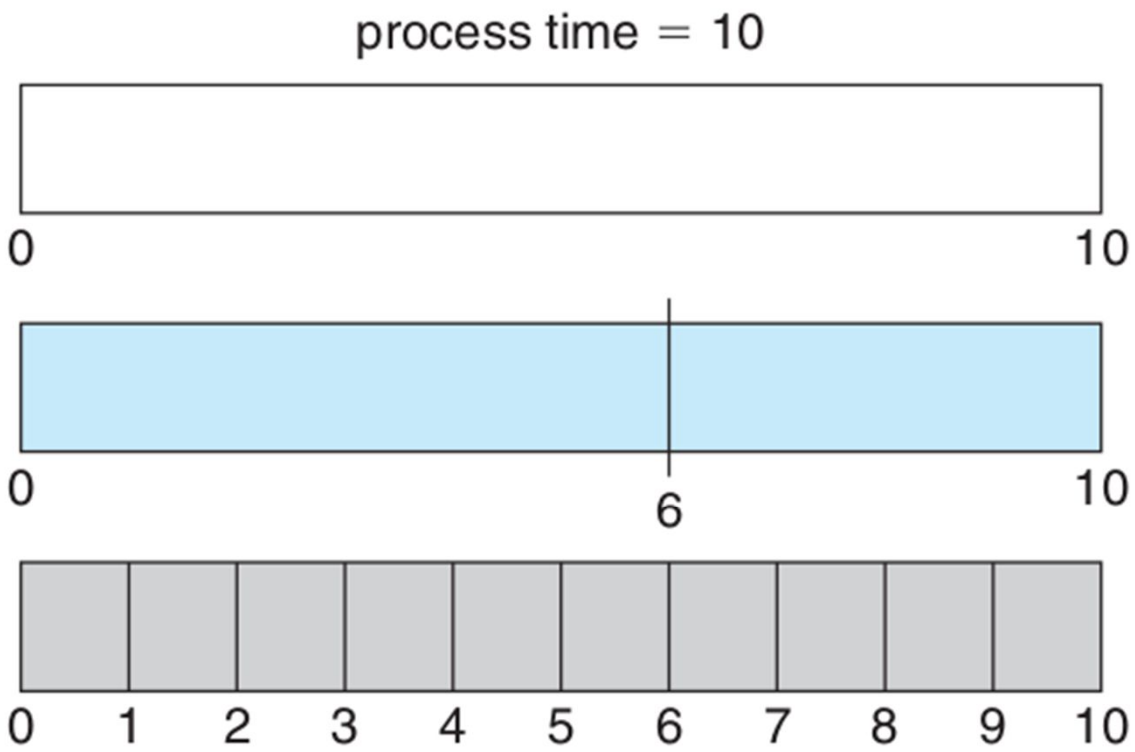
- Quantum = 4 ms



Impact of the quantum value in RR

- Too low a quantum causes too many process switches and lowers processor efficiency.
- Too high a quantum increases the response time for short commands in interactive mode.
- A quantum between 20 and 50 ms is often a reasonable compromise.

Cont'



quantum	context switches
12	0
6	1
1	9

Preemptive schedulers: Exercise II

- Let two processes A and B be ready such that A arrived first followed by B, 2 time units later. The processor times required to execute processes A and B are 15 and 4 time units respectively. Switching time is assumed to be zero.
  - *Calculate the stay time of each process A and B, the average stay time, the wait time, the average wait time, and the number of context switches for:*
  - *SRT*
    - *Round robin (quantum = 10 time units)*
    - *Round robin (quantum = 3 units of time)*

Correction : SNRT

Solution SRT :

A	A	B	B	B	B	A	A	A	A	A	A	A	A	A	A	A	A	A
1				5					10					15				

Process	Arrival time	Execution time	Stay time	Waiting time
P_1	0	15	19-0 = 19	19-15 = 4
P_2	2	4	6-2 = 4	4 – 4 = 0

- Average stay time: **(19+4)/2 = 11.5 ms**
- Average waiting time: **(4+0)/2 = 2 ms**
- Number of context switches: **2**

Exercise II : Correction RR quantum 10

Round robin (quantum = 10) :

A	A	A	A	A	A	A	A	A	A	B	B	B	B	A	A	A	A	A
1				5					10					15				

Processus	Tps d'arrivée	Tps d'exécution	Tps de séjour= Terminaison - Arrivée	Tps d'attente= Séjour - Exécution
A	0	15	19-0 = 19	19-15 = 4
B	2	4	14-2 = 12	12 – 4 = 8

- Temps moyen de séjour : **(19+12)/2 = 15.5 ms**
- Temps moyen d'attente : **(4+8)/2 = 6 ms**
- Le nombre de changements de contexte **: 2**

Exercise II : Correction RR quantum 3

Round robin (quantum = 3) :

A	A	A	B	B	B	A	A	A	B	A	A	A	A	A	A	A	A	A
1				5					10					15				

Process	Arrival time	Execution time	Stay time	Waiting time
A	0	15	19-0 = 19	19-15 = 4
B	2	4	10-2 = 8	8 – 4 = 4

- Average stay time: **(19+8)/2 = 13.5 ms**
- Average waiting time: **(4+4)/2 = 4 ms**
- Number of context switches: **4**

- Consider five processes P_1, P_2, P_3, P_4, and P_5, whose respective execution times and arrival times are as follows:

Process	*Arrival time*	*Execution time*
P_1	0	5
P_2	1	3
P_3	3	6
P_4	5	1
P_5	6	4

- Switching time is assumed to be zero. Draw the GANTT diagram for Round Robin scheduling (quantum = 3 units of time)

Correction of exercise no. III

Process	Arrival time	Execution time
P_1	0	5→ 2→ 0
P_2	1	3→ 0
P_3	3	6→ 3→ 0
P_4	5	1→0
P_5	6	4→ 1→ 0

P_1
P_2
P_3
P_1
P_4
P_5
P_3
P_5

Waiting queue

P_1	P_2	P_3	P_1	P_4	P_5	P_3	P_5
3	6	9	11	12	15	18	19

Exercise IV – Round Robin

- Consider five processes A, B, C, D and E whose respective execution times and arrival times are as follows:

Process	*Arrival time*	*Execution time*
A	0	3
B	2	6
C	4	4
D	6	5
E	8	2

- Switching time is assumed to be zero. Draw the GANTT diagram for Round Robin scheduling (quantum = 1 time units then quantum = 3).

Correction : Exercise n° IV (Quantum=1)

Process	Arrival time	Execution time
A	0	3→2 → 1→0
B	2	6→5 →4 → 3 → 2 →1 →0
C	4	6→ 3→ 0
D	6	1→0
E	8	4→ 1→ 0

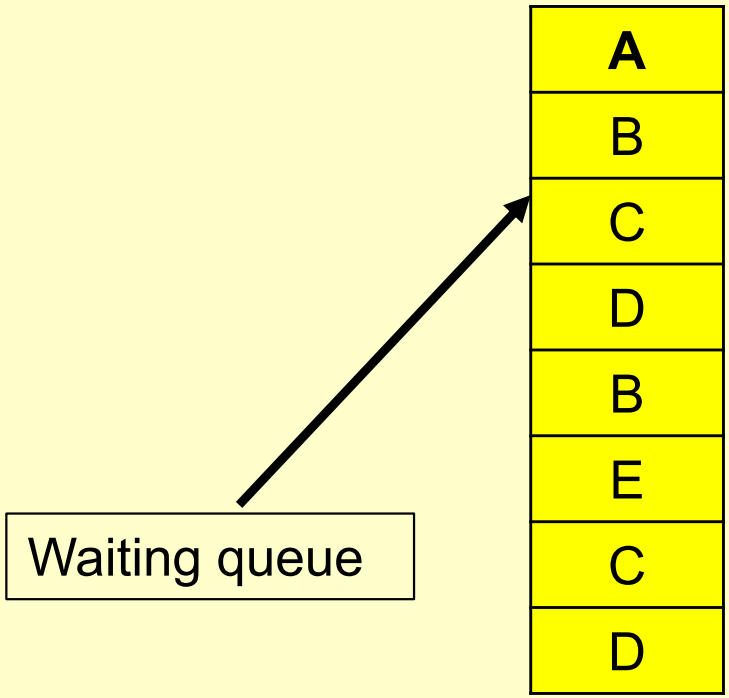
1	A	11	E
2	A	12	D
3	B	13	C
4	A	14	B
5	B	15	E
6	C	16	D
7	B	17	C
8	D	18	B
9	C	19	D
10	B	20	D

Waiting queue

A	A	B	A	B	C	B	D	C	B	E	D	C	B	E	D	C	B	D	D
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20

Correction : Exercise n° IV (Quantum=3)

Process	Arrival time	Execution time
A	0	**3 → 0**
B	2	**6 → 3 → 0**
C	4	**4 → 1 → 0**
D	6	**5 → 2 → 0**
E	8	**2 → 0**



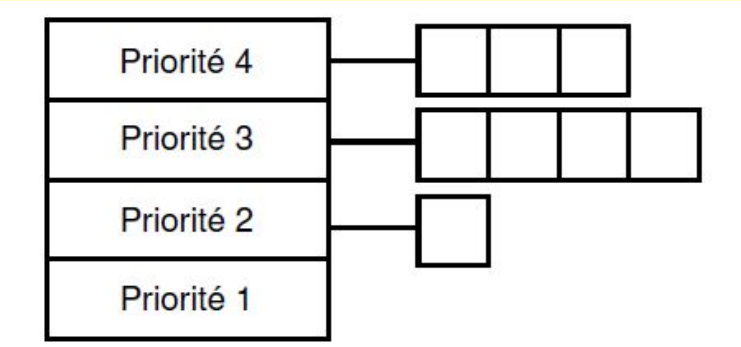
A			B			C			D			B			E		C	D	
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20

Impact of quantum in RR

- If the quantum is too large, we revert to the FIFO algorithm.
- If the quantum is too short, there is a drop in the useful rate (too many context changes).

Allocation and evolution of priorities

- To prevent high-priority processes from running indefinitely, the scheduler regularly reduces the priority of the process currently running.
- The priority of the current process is regularly compared with that of the highest priority ready process (at the head of the queue).
  - *When the priority falls below that of the highest-priority ready process (at the head of the queue), the process is switched.*
- In this case, the suspended process is inserted into the queue with its new priority.
- The allocation and evolution of priorities depend on the objectives set and many other parameters.
- Processes are classified by priority in different queues



Evolution of the priority

- Algorithms using a fixed priority are likely to lead to starvation of low-priority processes.
  - *The priority is changed according to the time spent away from the processor.*
  - *For example, in BSD (Berkeley Software Distribution: an operating system derived from Unix), a new priority is recalculated every n clock times:*

$$P_{real} = P_{min} + \frac{T_{cpu}}{n} + 2 * P_{nice}$$

where :

- P_{real} *is the calculated priority (a value between* P_{min} *and* P_{max} *),*
- P_{min} *is the maximum priority,*
- P_{max} *is the minimum priority,*
- T_{cpu} *is an estimate of the time spent in the processor,*
- P_{nice} *is a value assigned by the system (good behaviour value).*

Process scheduling under LINUX

Process states

- Reports can be examined using :
  - *the command ps or*
  - *by looking at the contents of the pseudo-file /proc/<pid>/status.*
    - *This file contains a line **State**:... indicating the state of the process.*

Runing (R)	the process is running, actively working
Sleeping (S)	the process is waiting for an external event. It places itself in a sleep state.
Stopped (T)	the process has been temporarily stopped by a signal. It will no longer run and will only react to a start signal.
Zombie (Z)	the process has been terminated, but his parent has not yet read his return code.

Multitasking, priorities
Several R's on a single-processor machine?

- At the level of the command “ps” or the pseudo-file /proc/<pid>/status , there is no difference between a process that is actually running and one that is ready.
  - *They are both indicated by the letter R.*
  - *This explains why the command “ps aux” sometimes presents a list containing several processes in state R at the same time on a single-processor machine.*

Multitasking and priorities

- An application that performs large ranges of computing operations without requesting I/O makes heavy use of the only resource that is really essential for all processes: the CPU.
- This application will therefore penalize other applications that make more reasonable use of the CPU.

Multitasking, priorities
Dual priorities

- To overcome this problem, the kernel uses the principle of double priority.
- A static priority value is given to each process as soon as it is started, and can be partially corrected by an appropriate system call.
- The scheduler uses this static value to calculate a new value, called dynamic priority, which is updated each time the process is processed by the scheduler.
  - *This dynamic priority is entirely internal to the kernel and cannot be modified.*

- The more CPU time a process uses, the lower the kernel's dynamic priority.
- On the contrary, the faster the process gives up when it is executed, the higher the kernel's priority.
  - *With this policy, tasks that make little use of the processor*
  - *trigger an I/O operation and immediately go to sleep waiting for the result*
  - *will go from the Ready state to Running state much more quickly than tasks that take up all the CPU cycles offered to th, without ever voluntarily giving up their hand.*

- It is necessary to be able to give the kernel an indication of the priority assigned to a particular process.
- When several processes are ready, the kernel will first choose the one with the highest dynamic priority.
  - *When calculating the dynamic priority, the scheduler uses the static priority in conjunction with other factors, such as whether the process has released the processor before its time limit has expired.*

- The higher the dynamic priority of a process, the longer the time slice allocated to it.
  - *This is a way of penalizing programs that loop, letting them work anyway, but without disturbing the other processes too much.*
- When a process has used up all its cycles, it will only be re-elected for processor access if no other, more courteous process is ready.

nice or courteous

- When a process wants to influence its own static priority, it can use the ***nice()*** system call.
- The declaration of this function can be found in <unistd.h> :

```
int nice (int increment);
```

- The value passed is added to our nice with respect to other processes.
  - *This means that a positive increment decreases the priority of the process, while a negative increment increases its priority.*

- The range of values used internally by the scheduler for static priorities is from 0 to 40.
- However, by convention, the kindness of a process is presented on a scale from -20 (very selfish process) to +19, with 0 being the default value.

- `$nice -n prio cmd :`
  - *launches the command "cmd" with priority prio.*
- `$renice -n prio -p PID :`
  - *changes the priority of the PID process.*
- `$renice -n prio -p USR:`
  - *changes the priority of all USR processes.*

Appendix: Scheduling in Linux and Windows

- The Linux system directly schedules threads (not processes).
- It defines several classes of threads:
  - *2 so-called "real-time" classes*
    - *FIFO (SCHED_FIFO)*
    - *Round Robin (SCHED_RR)*

    which are not really real-time, because they do not provide any guarantees about deadlines.
  - *Time-sharing threads:*
    - *Standard threads*
    - *(SCHED_OTHER) Calculation threads*
    - *(SCHED_BATCH) Background task threads (SCHED_IDLE)*

- Real-time threads
  - *They always have a higher priority than timesharing threads.*
  - *They have a static priority of between 0 and 99.*
  - *They are reserved for the system.*
  - *The difference between the two is :*
    - *FIFO threads can only be interrupted by a higher priority thread.*
    - *RR threads can be interrupted by a higher priority thread or when their time quantum is reached.*

- Time-sharing threads
  - *SCHED_OTHER normal threads:*
    - *Their static priority is zero (they are always slower than real-time threads),*
    - *Their time quantum is smaller,*
    - *They have a dynamic priority based on a courtesy value (nice) which changes according to their processor usage,*
    - *The system applies a bonus to threads that block (on input/output) and a penalty to processes that use the processor.*

      $\Rightarrow$ *interactive processes are favored.*
    - *In all, there are 40 priorities for time-sharing threads.*

- *Batch processing threads (SCHED_BATCH):*
  - *identical to the previous ones,*
  - *but are necessarily considered as threads using the processor, and are therefore penalized.*
- *Background tasks :*
  - *without static priority or courtesy,*
  - *scheduled last.*

Windows

- The system offers 32 priorities:
  - *from 0 to 15 for user threads*
  - *from 16 to 31 for the system (Real Time)*
- To decide between threads with the same priority, the system uses Round Robin.
- The priority is calculated in 2 parts:
  - *Basic process priority: IDLE_PRIORITY_CLASS, BELOW_NORMAL_PRIORITY_CLASS, NORMAL_PRIORITY_CLASS, ABOVE_NORMAL_PRIORITY_CLASS, HIGH_PRIORITY_CLASS, REALTIME_PRIORITY_CLASS.*
  - *Thread priority in the process: THREAD_PRIORITY_IDLE, THREAD_PRIORITY_LOWEST, THREAD_PRIORITY_BELOW_NORMAL, THREAD_PRIORITY_NORMAL, THREAD_PRIORITY_ABOVE_NORMAL, THREAD_PRIORITY_HIGHEST, THREAD_PRIORITY_TIME_CRITICAL,*

	PROCESS					
	Idle	Below n.	Normal	Above n.	High	R. T
Idle	1	1	1	1	1	16
Lowest	2	4	6	8	11	22
Below n.	3	5	7	9	12	23
Normal	4	6	8	10	13	24
Above n.	5	7	9	11	14	25
Highest	6	8	10	12	15	26
Time Crit.	15	15	15	15	15	31

- Bonus/penalty
  - *bonus for any process whose window comes to the foreground,*
  - *bonus for processes that wake up following an I/O,*
  - *penalty for a thread that finishes its quantum of time,*
  - *bonus for a thread that has not had a hand for "too long".*
- Priority limits:
  - *THREAD_PRIORITY_IDLE: 16 for real times and 1 for normal times,*
  - *THREAD_PRIORITY_TIME_CRITICAL: 31 for real times and 15 for normal times.*