# CHAPTER 4- PROCESS SCHEDULING

**Recall the process state diagram:**

|  |  |
| --- | --- |
|  | **Legend:**   1. Process blocks to wait for I/O or to sleep 2. Interrupt: scheduler picks another process 3. Scheduler picks this process to execute 4. I/O finished for this process, or it wakes up |

**Scheduler:**

The short term scheduler selects the process to get the processor (cpu) from among the processes which are already in memory.(i.e. from the ready queue). This scheduler will be executing frequently. So it has to be very fast in order to achieve a better processor utilization.

**Processor Scheduling Algorithms:**

Processor scheduling algorithms try to answer the following crucial question. Which process in the ready queue will get the processor­? In order to answer this, one should consider the relative importance of the following performance criteria.

**Performance Criteria:**

1. **Processor utilization:**= (processor busy time) / (processor busy time + processor idle time)

We would like to keep the processor as busy as possible, i.e. we want to reduce the idle time, (in real life, 40% - 90%).

1. **Throughput:**

= no. of processes completed per time unit

→ It is the measure of work done in a unit time interval.

1. **Turnaround time (tat) :**

= The interval from the time of submission to the time of completion of a process.

→ It is the sum of: time spent waiting to get into the ready queue, execution time and I/O time.

1. **Waiting time :**

= The average time spent time by a process waiting in the ready queue.

The processor scheduling algorithm only affects this time.

→ Considering only the waiting time for each process instead of the **tat** will be sufficient.

1. **Response time: (for interactive systems)**

= The time from the submission of request until getting the first response.

It is desirable that:

|  |  |  |
| --- | --- | --- |
| Process utilization | → | Max. |
| Throughput | → | Max. |
| Turnaround | → | Min. |
| Waiting time | → | Min. |
| Response time | → | Min. |

**Preemptive versus Non-preemptive Scheduling:**

A scheduling decision may take place under the five possible circumstances:

1. After a process switches from the running state to the blocked (waiting) state (process requests an I/O).
2. After a process switches from the running state to the ready state (interrupt occurs).
3. After a process switches from the blocked (waiting) state to the ready state (completion of I/O).
4. After a process terminates (completion of the running process).
5. After a newly created process arrives at ready state (a new job is accepted).

For circumstances 1 and 4 → a new process must be selected from the ready processes queue.

For circumstances 2, 3 and 5 → should the previously running process be re-selected or would it be possible to select a new process from the ready processes queue?

* A scheduling algorithm which acts only on circumstances 1 and 4 is called non-preemptive. (i.e. the algorithm will reselect the previously running process in circumstances 2, 3 and 5).

→ Once CPU has been allocated to a process, that process keeps the CPU until it release the CPU (either by termination or by requesting I/O)

* A scheduling algorithm which acts on all circumstances is called preemptive. (i.e. such an algorithm can select a new process in circumstances 2, 3 and 5).

→ The highest-priority process among all ready processes is allocated the CPU. The scheduler is called each time a process enters the ready state.

Advantages of non-preemptive scheduling algorithms are :

* + They cannot lead the system to a race condition.
  + They are simple.

Disadvantage of non-preemptive scheduling algorithms is:

* + They do not allow real multiprogramming.

Advantage of preemptive scheduling algorithms is:

* + They allow real multiprogramming.

Disadvantages of preemptive scheduling algorithms are:

* + They can lead the system to a race condition,
  + They are complex.

## 1. FCFS Scheduling Algorithm (non-preemptive)

**The First Come First Served (FCFS) Scheduling Algorithm** is the simplest one. In this algorithm the set of ready processes is managed as FIFO (first-in-first-out) Queue. The processes are serviced by the CPU until completion in order of their entering in the FIFO queue. A process once allocated the CPU keeps it until releasing the CPU either by terminating or requesting I/O. For example, interrupted process is allowed to continujre running after interrupt handling is done with.

*Example:*

Let processes P1, P2, and P3 arrive to the ready queue in that order and let the run times (CPU burst times) of these processes be as follows:

|  |  |
| --- | --- |
| **Process** | **CPU burst** |
| **P1** | **24** |
| **P2** | **3** |
| **P2** | **3** |

These processes are served by the CPU in FCFS order. The result can be shown in the following Gantt chart:

|  |  |  |
| --- | --- | --- |
| **P1** | **P2** | **P3** |

**0 24 27 30**

The waiting time for P1 is: **W (P1) = 0 ms**

The waiting time for P2 is**: W (P2) = 24 ms**

The waiting time for P3 is: **W (P3) = 27 ms**

**→ The average waiting time is: W = (0+24+27)/3 = 17 ms.**

If these processes arrived in the following order: P2, P3, P1 then the Gantt chart would be as follows:

|  |  |  |
| --- | --- | --- |
| **P2** | **P3** | **P1** |

**0 3 6 30**

**The average waiting time is now: W = (0+3+6)/3 = 3 ms**

*Exercise:*

Consider the following processes with the relative CPU bursts which arrive together in the order given.

|  |  |
| --- | --- |
| **Process** | **CPU burst** |
| **A** | **16** |
| **B** | **4** |
| **C** | **1** |

Draw the Gantt charts, and find out the average turn around time (**tat**) for FCFS CPU scheduling algorithm if:

1. Coming order is A, B, C.
2. Coming order is C, B, A.

## 2. SJF Scheduling Algorithm (non-preemptive)

**The Shortest Job First Scheduling** Algorithm chooses the process that has the smallest next CPU burst.

*Example:*

Assume there are 4 ready processes with their next CPU burst time as follows:

|  |  |
| --- | --- |
| **Process** | **Next CPU burst time** |
| **P1** | **6** |
| **P2** | **8** |
| **P3** | **7** |
| **P4** | **3** |

Using SJF scheduling algorithm, these processes are scheduled according to the following Gantt Chart:

|  |  |  |  |
| --- | --- | --- | --- |
| **P4** | **P1** | **P3** | **P2** |

**0 3 9 16 24**

**So, the average waiting time is (3+16+9+0) / 4 =7 ms**

NOTE:

If the processes (in order P1, P2, P3, P4) are scheduled using the FCFS algorithm then the average waiting time is (0+6+14+21) /4 = 10.25 ms.

|  |  |  |  |
| --- | --- | --- | --- |
| **P1** | **P2** | **P3** | **P4** |

**0 6 14 21 24**

→ SJF minimizes the average wait time because it services smaller processes before large ones.

* The main problem with the SJF algorithm is to know for each process the next CPU burst time!!! However some techniques exist to predict this next CPU burst time.
* Another problem is that starvation of long processes may occur.

*Exercise:*

Consider the following processes which are in the ready queue in the given order with the relative next CPU bursts.

|  |  |
| --- | --- |
| **Process** | **Next CPU burst time** |
| **P1** | **7** |
| **P2** | **14** |
| **P3** | **3** |
| **P4** | **6** |

Draw the Gantt charts, fill in the following table:

|  |  |  |
| --- | --- | --- |
|  | **Average tat** | **Average W** |
| **FCFS scheduling** |  |  |
| **SJF scheduling** |  |  |

**FCFS:**

|  |  |  |  |
| --- | --- | --- | --- |
| **P1** | **P2** | **P3** | **P4** |

**0 7 21 24 30**

**W (P1) = 0 W (P2) = 7 W (P3) = 21 W (P4) = 24**

**tat (P1) = 7 tat (P2) = 21 tat (P3) = 24 tat (P4) =30**

**Average W = (21+7+24)/4 = 13 ms**

**Average tat = (7+21+24+30)/4 = 20.5 ms**

**SJF:**

|  |  |  |  |
| --- | --- | --- | --- |
| **P3** | **P4** | **P1** | **P2** |

**0 3 9 16 30**

**W (P1) = 9 W (P2) = 16 W (P3) = 0 W (P4) = 3**

**tat (P1) = 16 tat (P2) = 3 tat (P3) = 3 tat (P4) =9**

**Average W = (9+16+3)/4 = 7 ms**

**Average tat = (16+30+3+9)/4 = 14.5 ms**

## 3. SRTF: Shortest Remaining Time First (preemptive)

This is the preemptive version of SJF. The currently executing process will be preempted from the CPU if a process with a shorter CPU burst time is arrived.

*Example:*

Consider the following four processes:

|  |  |  |
| --- | --- | --- |
| **Process** | **Arrival time** | **CPU burst time** |
| **P1** | **0** | **8** |
| **P2** | **1** | **4** |
| **P3** | **2** | **9** |
| **P4** | **3** | **5** |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **P1** | **P2** | **P4** | **P1** | **P3** |

**0 1 5 10 17 26**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **W (P1) = 10-1 = 9 ms**  **W (P2) = 0 ms**  **W (P3) = 17-2 = 15 ms**  **W (P4) = 5-3 = 2 ms**  **Avg. waiting time:**  **= (9+0+15+2) / 4**  **= 26/4**  **= 6.5 ms** | **tat (P1) = 17 ms**  **tat (P2) = 5-1 = 4 ms**  **tat (P3) = 26-2 = 24 ms**  **tat (P4) = 10-3 = 7 ms**  **Average tat = 13 ms** | **How to process preemptive scheduling:**   |  |  |  |  |  | | --- | --- | --- | --- | --- | | **time** | **P1** | **P2** | **P3** | **P4** | | **0** | **8** |  |  |  | | **1** | **7** | **4** |  |  | | **2** | **7** | **3** | **9** |  | | **3** | **7** | **2** | **9** | **5** | | **…** |  |  |  |  | |

If the SJF were used which is non-preemptive:

|  |  |  |  |
| --- | --- | --- | --- |
| **P1** | **P2** | **P4** | **P3** |

**0 8 12 17 26**

**W (P1) = 0 ms (P2) = 8-1 = 7 ms**

**W (P3) = 17-2 = 15 ms (P4) = 12-3= 9 ms**

**Average waiting time = (0+7+15+9) /4 = 31/4 = 7.75 ms**

=> SJF gives smaller wait time than FCFS since shorter processes are executed before longer ones!

*Exercise:*

Consider the following four processes

|  |  |  |
| --- | --- | --- |
| **Process** | **Arrival time** | **CPU burst time** |
| **P1** | **0** | **15** |
| **P2** | **2** | **2** |
| **P3** | **2** | **9** |
| **P4** | **10** | **3** |

Draw the Gantt charts and fill in the following table:

|  |  |  |
| --- | --- | --- |
|  | **Average tat** | **Average W** |
| **SJF scheduling** |  |  |
| **SRTF scheduling** |  |  |

## 4. Round Robin Scheduling (preemptive)

* This scheduling algorithm is designed specially for time sharing systems. It is similar to FCFS scheduling, but preemption is added to switch between processes.
* A small unit of time, called a *time quantum*, or time slice, is assigned to each process. Usually a quantum is 10 to 100 ms.
* This scheduler allocates the CPU to each process in the ready queue for a time interval of up to 1 time quantum in FIFO (circular) fashion. If the process still running at the end of the quantum; it will be preempted from the CPU. A context switch will be executed, and the process will be put at the tail of the ready queue. Then the scheduler will select the next process in the ready queue. However, if the process has blocked or finished before the quantum has elapsed, the scheduler will then proceed with the next process in the ready queue.

*Example:*

Consider the following set of process that arrive at time 0 with the order A,B,C,D and the following CPU burst time. Find the average waiting time with RR of quantum : 10 ms

|  |  |
| --- | --- |
| **Process** | **CPU burst time** |
| **A** | **20** |
| **B** | **40** |
| **C** | **14** |
| **D** | **6** |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **A** | **B** | **C** | **D** | **A** | **B** | **C** | **B** | **B** |

**0 10 20 30 36 46 56 60 70 80**

**W (A) = 36-10 =26 ms W (B) = (10-0) + (46-20) + (60-56)= 40 ms**

**W (C) = (20-0) + (56-30) = 46 ms W (D) = 30-0 = 30 ms**

**Average waiting time = (26+40+46+30) /4 = 142/4 = 35.5 ms**

*Example:*

Redo the previous example by introducing switching time of 2 ms and some I/O. i.e. A has 10 ms CPU burst – 40 ms I/O burst -10 ms CPU burst , and the others are as before.

**CPU:**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **A1** |  | **B1** |  | **C1** |  | **D1** |  | **B2** |  | **C2** |  | **A2** |  | **B3** |  | **B4** |

**0 10 12 22 24 34 36 42 44 54 56 60 62 72 74 84 86 96**

**I/O:**

|  |  |  |
| --- | --- | --- |
| **idle** | **A** | **idle** |

**0 10 50**

**Wait (A) = 62-50 = 12 ms Wait (B) = 12+22+20+2 = 56 ms**

**Wait (C) = 24 + 22 = 46 ms Wait (D) = 36 ms**

**Average waiting time = 150/4 = 37.5 ms**

**Important note:** smaller time quantum increases the number of context switches! So, time quantum should be large with respect to the context switching time.

**Note:** if time of quantum = ∞, =>RR becomes FCFS.

**Example: Process scheduling with I/O:**

Consider the following process arrival CPU, I/O. burst times given. Assume that context switching time negligible and there is a single I/O device which operates in FCFS manner.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Process** | **Arrival time** | **CPU-1** | **I/O-1** | **CPU-2** | **I/O-2** | **CPU-3** | **I/O-3** |
| **A** | **0** | **5** | **10** | **5** | **10** | **5** | **10** |
| **B** | **5** | **5** | **12** | **5** | **12** | **5** | **12** |

Draw the Gantt charts for the CPU and I/O device, and find out the average turn around time (tat) and utilization for FCFS processor scheduling algorithm.

**CPU:**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **A1** | **B1** |  | **A2** |  | **B2** |  | **A3** |  | **B3** |

**0 5 10 15 20 27 32 37 42 49 54**

**I/O:**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **A1** | **B1** | **A2** | **B2** | **A3** | **B3** |

**0 5 15 27 37 49 59 71**

**tat(a) = 59 ms**

**tat(b) = 66 ms**

**Avg. tat = (59+66)/2 = 62.5 ms**

**CPU Utilization = 30/54 x 100 % = 55.55 %**

## 5. Priority scheduling (preemptive and non-preemptive)

The basic principle of the priority scheduling algorithm is to assign a priority order to each ready process. The ready process with the highest priority is allowed to run. Equal priority processes are scheduled randomly. Priority scheduling can be of:

* Non-preemptive ( a process runs up to the end of its CPU burst time)
* Preemptive (a running process can be preempted can be preempted by a process with a higher priority which joined the ready queue later)

Example:

|  |  |  |  |
| --- | --- | --- | --- |
| **Process** | **Arrival time** | **CPU burst time** | **Priority** |
| **A** | **0** | **100** | **3** |
| **B** | **0** | **10** | **1** |
| **C** | **0** | **300** | **3** |
| **D** | **0** | **60** | **5** |
| **E** | **80** | **150** | **4** |

Draw the Gantt chart and compute the average waiting time in case of:

1. Non-preemptive
2. Preemptive.
3. Non-preemptive:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **D** | **A** | **E** | **C** | **B** |

**0 60 160 310 610 620**

**W (A) = 60 ms W (B) = 610 ms**

**W (C) = 310ms W (D) = 0 ms**

**W (E) = 80 ms**

**=> Average waiting time: 212 ms**

1. Preemptive:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **D** | **A** | **E** | **A** | **C** | **B** |

**0 60 80 230 310 610 620**

**Wait (A) = 60+150 = 210 ms Wait (B) = 610 ms**

**Wait (C) = 310 ms Wait (D) = 0 ms**

**Wait (E) = 0 ms**

**=> Average waiting time: 226 ms**

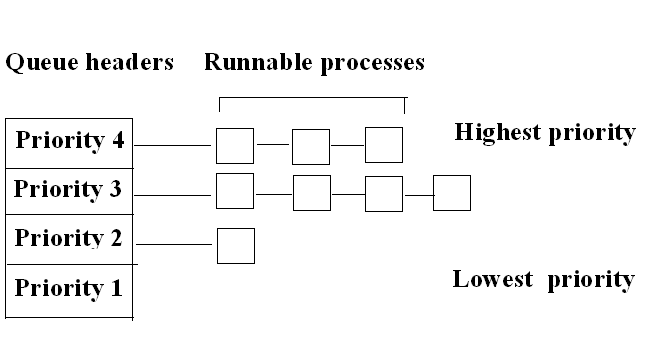
**Note:** SJF is a case of the priority scheduling where priority = 1/ (next CPU burst time).

* A major problem with the priority scheduling is **starvation**: a low priority process may wait for ever before being executed. ( MIT IBM 7094 crashed in 1973. They found a ready process submitted in 1967 and still waiting for execution).

*Solution*: Gradually increase the priority of processes that wait for a long time (this technique is called **aging**).

* The priority scheduling algorithm may be combined with the RR algorithm. In this case, processes are grouped into priority classes:
* Among classes a priority scheduling algorithm is used.
* Inside a class a RR algorithm is used => 1 ready queue for each class.

The following shows a system with four priority classes. The scheduling algorithm is as follows: as long as there are runnable processes in priority class 4, just run each one for one quantum(i.e. round robin fashion), and never bother with lower priority classes. If priority class 4 in empty , then run the class 3 processes in round robin fashion , and so on. If priorities are not adjusted from time to time, lower priority classes may all starve.

****

**Fig.** A scheduling algorithm with four priority classes.

*Example:*

|  |  |  |  |
| --- | --- | --- | --- |
| **Process** | **Arrival time** | **CPU burst time** | **Priority** |
| **A** | **0** | **22** | **1** |
| **B** | **0** | **12** | **2** |
| **C** | **0** | **23** | **2** |
| **D** | **0** | **11** | **3** |
| **E** | **0** | **21** | **2** |
| **F** | **0** | **15** | **3** |
| **G** | **30** | **22** | **3** |

Assume quantum =10 ms , and 0 switching time, draw the Gantt chart and compute the average waiting time in case of:

1. Non-preemptive priority scheduling
2. Preemptive priority scheduling

Queue\_1 : A

Queue\_2 : B, C, E

Queue\_3 : D, F, G (Join the queue at time 30)

**a)**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **D1** | **F1** | **D2** | **F2** | **B1** | **C1** | **E1** | **B2** | **C2** | **E2** | **C3** | **E3** | **G1** | **G2** | **G3** | **A1** | **A2** | **A3** |

**0 10 20 21 26 36 46 56 58 68 78 81 82 92 102 104 114 124**

**W (A) = 104 W (B) = 46 W (C) =58 W (D) = 10**

**W (E) =61 W(F) =11 W (G) =52 => Average W: 48.8**

**b)**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **D1** | **F1** | **D2** | **F2** | **B1** | **G1** | **G2** | **G3** | **C1** | **E1** | **B2** | **C2** | **E2** | **C3** | **E3** | **A1** | **A2** | **A3** |

**0 10 20 21 26 36 46 56 58 68 78 80 90 100 103 104 114 124**

**W (A) = 104 W (B) = 68 W (C) =80 W (D) = 10**

**W (E) =83 W (F) =11 W (G) =6 => Average W = 51.7**