Homework 05

CS307-Operating System (D), Chentao Wu, Spring 2020.

Name: 方泓杰(Hongjie Fang) Student ID: 518030910150 Email: galaxies@sjtu.edu.cn

- (5.4) Consider the following set of processes, with the length of the CPU burst time given in milliseconds:

Process	Burst Time	Priority
P_1	2	2
P_2	1	1
P_3	8	4
P_4	4	2
P_5	5	3

The processes are assumed to have arrived in the order P_1, P_2, P_3, P_4, P_5, all at time 0.

a. Draw four Gantt charts that illustrate the execution of these processes using the following scheduling algorithms: FCFS, SJF, non-preemptive priority (a larger priority number implies a higher priority), and RR (quantum = 2).

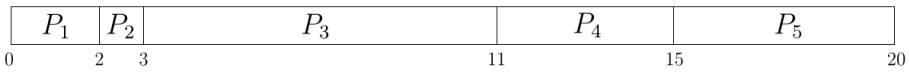
b. What is the turnaround time of each process for each of the scheduling algorithms in part a?

c. What is the waiting time of each process for each of these scheduling algorithms?

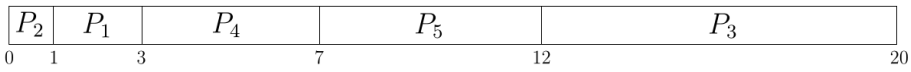
d. Which of the algorithms results in the minimum average waiting time (over all processes)?

Solution. Here are my solutions. *Default unit: millisecond (ms).*

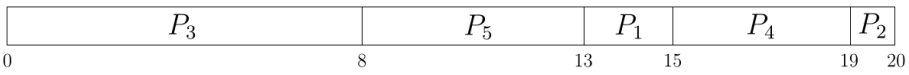
a. The four Gantt charts are as follows (Fig. 1).



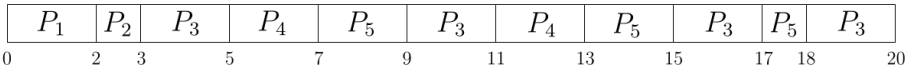
(a) Gantt chart of FCFS algorithm



(b) Gantt chart of SJF algorithm



(c) Gantt chart of non-preemptive priority algorithm



(d) Gantt chart of RR algorithm (with quantum = 2)

Figure 1: Gantt charts of four different algorithms

b. According to Gantt charts in Fig. 1, we can calculate the turnaround time of each process for each of the scheduling algorithms. Let tT_i denote the turnaround time of process P_i.

– **(FCFS)** According to Fig.1(a), we know:

$$tT_1 = 2 - 0 = 2, \qquad tT_2 = 3 - 0 = 3,$$
$$tT_3 = 11 - 0 = 11, \qquad tT_4 = 15 - 0 = 15,$$
$$tT_5 = 20 - 0 = 20.$$

– **(SJF)** According to Fig.1(b), we know:

$$tT_1 = 3 - 0 = 3, \qquad tT_2 = 1 - 0 = 1,$$
$$tT_3 = 20 - 0 = 20, \qquad tT_4 = 7 - 0 = 7,$$
$$tT_5 = 12 - 0 = 12.$$

– **(non-preemptive priority)** According to Fig.1(c), we know:

$$tT_1 = 15 - 0 = 15, \qquad tT_2 = 20 - 0 = 20,$$
$$tT_3 = 8 - 0 = 8, \qquad tT_4 = 19 - 0 = 19,$$
$$tT_5 = 13 - 0 = 13.$$

– **(RR with quantum = 2)** According to Fig.1(d), we know:

$$tT_1 = 2 - 0 = 2, \qquad tT_2 = 3 - 0 = 3,$$
$$tT_3 = 20 - 0 = 20, \qquad tT_4 = 13 - 0 = 13,$$
$$tT_5 = 12 - 0 = 18.$$

c. According to Gantt charts in Fig. 1, we can calculate the waiting time of each process for each of the scheduling algorithms. Let tW_i denote the waiting time of process P_i.

– **(FCFS)** According to Fig.1(a), we know:

$$tW_1 = 0 - 0 = 0, \qquad tW_2 = 2 - 0 = 2,$$
$$tW_3 = 3 - 0 = 3, \qquad tW_4 = 11 - 0 = 11,$$
$$tW_5 = 15 - 0 = 15.$$

– **(SJF)** According to Fig.1(b), we know:

$$tW_1 = 1 - 0 = 1, \qquad tW_2 = 0 - 0 = 0,$$
$$tW_3 = 12 - 0 = 12, \qquad tW_4 = 3 - 0 = 3,$$
$$tW_5 = 7 - 0 = 7.$$

– **(non-preemptive priority)** According to Fig.1(c), we know:

$$tW_1 = 13 - 0 = 13, \qquad tW_2 = 19 - 0 = 19,$$
$$tW_3 = 0 - 0 = 0, \qquad tW_4 = 15 - 0 = 15,$$
$$tW_5 = 8 - 0 = 8.$$

– **(RR with quantum = 2)** According to Fig.1(d), we know:

$$tW_1 = 0 - 0 = 0,$$
$$tW_2 = 2 - 0 = 2,$$
$$tW_3 = (3 - 0) + (9 - 5) + (15 - 11) + (18 - 17) = 12,$$
$$tW_4 = (5 - 0) + (11 - 7) = 9,$$
$$tW_5 = (7 - 0) + (13 - 9) + (17 - 15) = 13.$$

d. In problem c, we have calculated the waiting time of each process for each of the scheduling algorithms. We can calculate the average waiting time for each algorithm now.

$$tW_{FCFS} = \frac{1}{5} \cdot (0 + 2 + 3 + 11 + 15) = 6.2,$$
$$tW_{SJF} = \frac{1}{5} \cdot (1 + 0 + 12 + 3 + 7) = 4.6,$$
$$tW_{priority} = \frac{1}{5} \cdot (13 + 19 + 0 + 15 + 8) = 11.0,$$
$$tW_{RR} = \frac{1}{5} \cdot (0 + 2 + 12 + 9 + 13) = 7.2.$$

Therefore, we know that $tW_{SJF} < tW_{FCFS} < tW_{RR} < tW_{priority}$. Thus, the **SJF algorithm** results in the minimum average waiting time (over all processes).

□

- (5.5) The following processes are being scheduled using a preemptive, round-robin scheduling algorithm.

Process	Priority	Burst	Arrival
P_1	40	20	0
P_2	30	25	25
P_3	30	25	30
P_4	35	15	60
P_5	5	10	100
P_6	10	10	105

Each process is assigned a numerical priority, with a higher number indicating a higher relative priority. In addition to the processes listed below, the system also has an **idle task** (which consumes no CPU resources and is identified as P_{idle}). This task has priority 0 and is scheduled whenever the system has no other available processes to run. The length of a time quantum is 10 units. If a process is preempted by a higher-priority process, the preempted process is placed at the end of the queue.

a. Show the scheduling order of the processes using a Gantt chart.

b. What is the turnaround time for each process?

c. What is the waiting time for each process?

d. What is the CPU utilization rate?

Solution. Here are my solutions.

a. The Gantt chart of the scheduling order of processes is as follows (Fig. 2).

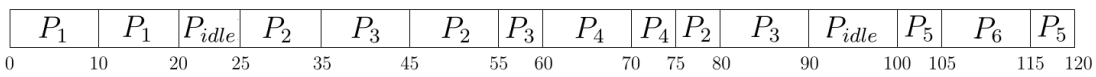


Figure 2: The Gantt chart of the scheduling order of processes

b. According to Fig. 2, we can calculate the turnaround time of each process as follows. Let tT_i denote the turnaround time of process P_i.

$$\begin{aligned} tT_1 &= 20 - 0 = 20, & tT_2 &= 80 - 25 = 55, \\ tT_3 &= 90 - 30 = 60, & tT_4 &= 75 - 60 = 15, \\ tT_5 &= 120 - 100 = 20, & tT_6 &= 115 - 105 = 10. \end{aligned}$$

c. According to Fig. 2, we can calculate the waiting time of each process as follows. Let tW_i denote the waiting time of process P_i.

$$\begin{aligned} tW_1 &= 0 - 0 = 0, \\ tW_2 &= (25 - 25) + (45 - 35) + (75 - 55) = 30, \\ tW_3 &= (35 - 30) + (55 - 45) + (80 - 60) = 35, \\ tW_4 &= 60 - 60 = 0, \\ tW_5 &= (100 - 100) + (115 - 105) = 10, \\ tW_6 &= 105 - 105 = 0. \end{aligned}$$

d. According to Fig. 2, we use 120 units of time to execute the processes totally. Only 15 units of time is used to execute P_{idle}, which consumes no CPU resources. Therefore, the CPU utilization rate can be calculated as follows (Eqn. (1)).

$$\text{CPU utilization rate} = \frac{120 - 15}{120} \times 100\% = 87.5\% \tag{1}$$

□

- (5.10) The traditional UNIX scheduler enforces an inverse relationship between priority numbers and priorities: the higher the number, the lower the priority. The scheduler recalculates process priorities once per second using the following function:

$$\text{Priority} = \frac{\text{recent CPU usage}}{2} + \text{base}$$

where base = 60 and recent CPU usage refers to a value indicating how often a process has used the CPU since priorities were last recalculated.

Assume that recent CPU usage for process P_1 is 40, for process P_2 is 18, and for process P_3 is 10. What will be the new priorities for these three processes when priorities are recalculated? Based on this information, does the traditional UNIX scheduler raise or lower the relative priority of a CPU-bound process?

Solution. Let p_i denote the new priority of process P_i. Then we can calculate p_i as follows.

$$\begin{aligned} p_1 &= \frac{\text{recent CPU usage}_{P_1}}{2} + \text{base} = \frac{40}{2} + 60 = 80 \\ p_2 &= \frac{\text{recent CPU usage}_{P_2}}{2} + \text{base} = \frac{18}{2} + 60 = 69 \\ p_3 &= \frac{\text{recent CPU usage}_{P_3}}{2} + \text{base} = \frac{10}{2} + 60 = 65 \end{aligned}$$

The original priority p_i' of each process P_i has a relation with its recent CPU usage. According to the CPU scheduling knowledge, we can assume they have the following relation (Eqn. (2)).

$$p_i' = k \cdot \text{recent CPU usage}_{P_i} + b \qquad (k, b \in \mathbb{R}, \quad k \geq 1) \tag{2}$$

Thus, the original relative priority of process P_i and P_j is $\left|p_i' - p_j'\right|$, and the current relative priority of process P_i and P_j is $|p_i - p_j|$. Hence,

$$\begin{aligned}
\frac{|p_i - p_j|}{\left|p_i' - p_j'\right|} &= \frac{\left|\left(\frac{1}{2} \cdot \text{recent CPU usage}_{P_i} + \text{base}\right) - \left(\frac{1}{2} \cdot \text{recent CPU usage}_{P_j} + \text{base}\right)\right|}{\left|\left(k \cdot \text{recent CPU usage}_{P_i} + b\right) - \left(k \cdot \text{recent CPU usage}_{P_j} + b\right)\right|} \\
&= \frac{1}{2k} \frac{\left|\text{recent CPU usage}_{P_i} - \text{recent CPU usage}_{P_j}\right|}{\left|\text{recent CPU usage}_{P_i} - \text{recent CPU usage}_{P_j}\right|} \\
&\leq \frac{1}{2}
\end{aligned} \tag{3}$$

Equation (3) tells us that the relative priority of process P_i and P_j is reduced by at least a half. Therefore, the traditional UNIX scheduler lower the relative priority of a CPU-bound process. □

- (5.18) The following processes are being scheduled using a preemptive, priority-based, round-robin scheduling algorithm.

Process	Priority	Burst	Arrival
P_1	8	15	0
P_2	3	20	0
P_3	4	20	20
P_4	4	20	25
P_5	5	5	45
P_6	5	15	55

Each process is assigned a numerical priority, with a higher number indicating a higher relative priority. The scheduler will execute the highest-priority process. For processes with the same priority, a round-robin scheduler will be used with a time quantum of 10 units. If a process is preempted by a higher-priority process, the preempted process is placed at the end of the queue.

a. Show the scheduling order of the processes using a Gantt chart.

b. What is the turnaround time for each process?

c. What is the waiting time for each process?

Solution. Here are my solutions.

a. The Gantt chart of the scheduling order of processes is as follows (Fig. 3).

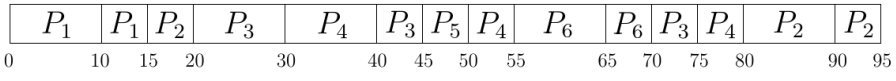


Figure 3: The Gantt chart of the scheduling order of processes

b. According to Fig. 3, we can calculate the turnaround time of each process as follows. Let tT_i denote the turnaround time of process P_i.

$$\begin{aligned} tT_1 &= 15 - 0 = 15, & tT_2 &= 95 - 0 = 95, \\ tT_3 &= 75 - 20 = 55, & tT_4 &= 80 - 25 = 55, \\ tT_5 &= 50 - 45 = 5, & tT_6 &= 70 - 55 = 15. \end{aligned}$$

c. According to Fig. 3, we can calculate the waiting time of each process as follows. Let tW_i denote the waiting time of process P_i.

$$\begin{aligned} tW_1 &= 0 - 0 = 0, \\ tW_2 &= (15 - 0) + (80 - 20) = 75, \\ tW_3 &= (20 - 20) + (40 - 30) + (70 - 45) = 35, \\ tW_4 &= (30 - 25) + (50 - 40) + (75 - 55) = 35, \\ tW_5 &= 45 - 45 = 0, \\ tW_6 &= 55 - 55 = 0. \end{aligned}$$

□

- (5.20) Which of the following scheduling algorithms could result in starvation?

a. First-come, first-served;

b. Shortest job first;

c. Round robin;

d. Priority.

Solution. Only **Shortest job first** and **Priority-based** scheduling algorithms could result in starvation. Here are the explanations.

a. First-come, first-served scheduling algorithm can guarantee that the processes are executed in their coming order. Thus, each of the processes will be executed when the processes coming before it all finish executing. Therefore, First-come, first-served scheduling algorithm cannot result in starvation.

b. Shortest-job first can result in starvation. Assume there is a process P_1 whose burst time is 20 ms, and there is another process P_2 coming every 10 ms, whose burst time is 10 ms. Then the CPU will be fully occupied with process P_2 in all time. Thus, process P_1 will never be executed, which indicates that Shortest-job-first scheduling algorithm can result in starvation.

c. Round-robin scheduling algorithm can guarantee that every process will be executed after the previous processes finish executing. Therefore, Round-robin scheduling algorithm cannot result in starvation.

d. Priority-based scheduling algorithm can result in starvation. Assume there are two processes P_1 and P_2 with burst time of 20 ms and 10 ms respectively. The priority of process P_2 is higher than the priority of process P_1, and the process P_2 will come every 10 ms. Then the CPU will be fully occupied with process P_2 in all time. Thus, process P_1 will never be executed, which indicates that Priority-based scheduling algorithm can result in starvation.

□