Student: Yu Cheng (Jade)

ICS 412

Homework #4

October 19. 2009

Homework #4

Exercise 5.12: Consider the following set of processes, with the length of the CPU burst given in milliseconds:

Process	*Burst Time*	*Priority*
P_1	10	3
P_2	1	1
P_3	2	3
P_4	1	4
P_5	5	2

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, nonpreemptive priority (a smaller priority number implies a higher priority), and RR (quantum = 1).

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 algorithm?

Answer: 1. FCFS:

```
-----------------------------------------------------------
|         P₁         | P₂ |     P₃    | P₄ |       P₅      |
-----------------------------------------------------------
0                   10    11         13    14             19
```

Performance statistics:

Process	*Arrival Time*	*Burst Time*	*Priority*	*Finish Time*	*Turnaround Time*	*Waiting Time*
P_1	0	10	3	10	10	0
P_2	0	1	1	11	11	10
P_3	0	2	3	13	13	11
P_4	0	1	4	14	14	13
P_5	0	5	2	19	19	14
Average					13.4	9.6

2. SJF (nonpreemptive):

```
-----------------------------------------------------------
| P2 | P4 |    P3    |     P5      |        P1         |
-----------------------------------------------------------
0     1    2          4             9                   19
```

Performance statistics:

Process	*Arrival Time*	*Burst Time*	*Priority*	*Finish Time*	*Turnaround Time*	*Waiting Time*
P_1	0	10	3	19	19	9
P_2	0	1	1	1	1	0
P_3	0	2	3	4	4	2
P_4	0	1	4	2	2	1
P_5	0	5	2	9	9	4
Average					7	3.2

3. SJF (preemptive, Shortest-next-CPU-burst):

In this case the processes all arrive at the same time. This method turned out to be the same as nonpreemptive SJF.

```
-----------------------------------------------------------
| P2 | P4 |    P3    |     P5      |        P1         |
-----------------------------------------------------------
0     1    2          4             9                   19
```

Performance statistics:

Process	*Arrival Time*	*Burst Time*	*Priority*	*Finish Time*	*Turnaround Time*	*Waiting Time*
P_1	0	10	3	19	19	9
P_2	0	1	1	1	1	0
P_3	0	2	3	4	4	2
P_4	0	1	4	2	2	1
P_5	0	5	2	9	9	4
Average					7	3.2

4. Nonpreemptive, Priority:

```
-----------------------------------------------------------
| P2 |      P5      |        P1         |   P3    | P4 |
-----------------------------------------------------------
0     1              6                   16        18    19
```

Performance statistics:

Process	*Arrival Time*	*Burst Time*	*Priority*	*Finish Time*	*Turnaround Time*	*Waiting Time*
P_1	0	10	3	16	16	6
P_2	0	1	1	1	1	0
P_3	0	2	3	18	18	16
P_4	0	1	4	19	19	18
P_5	0	5	2	6	6	1
Average					13.4	8.2

5. Round Robin:

```
----------------------------------------------------------------------------------
| P1 | P2 | P3 | P4 | P5 | P1 | P3 | P5 | P1 | P5 | P1 | P5 | P1 | P5 | P1 | P1 | P1 | P1 | P1 |
----------------------------------------------------------------------------------
0    1    2    3    4    5    6    7    8    9    10   11   12   13   14   15   16   17   18   19
```

Performance statistics:

Process	*Arrival Time*	*Burst Time*	*Priority*	*Finish Time*	*Turnaround Time*	*Waiting Time*
P_1	0	10	3	19	19	9
P_2	0	1	1	2	2	1
P_3	0	2	3	7	7	5
P_4	0	1	4	4	4	3
P_5	0	5	2	14	14	9
Average					9.2	5.4

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

Answer: Average Waiting Time:

Algorithm	*Average Waiting Time*
FCFS	9.6
SJF (nonpreemptive)	3.2
SJF (preemptive, Shortest-next-CPU-burst)	3.2
Nonpreemptive, Priority	8.2
Round Robin	5.4

Therefore, SJF (either nonpreemptive algorithm or preemptive algorithm, since they given the same result in this case) has the minimum average waiting time.

Exercise 5.15: Consider a system running ten I/O-bound tasks and one CPU-bound task. Assume that the I/O-bound tasks issue an I/O operation once for every millisecond of CPU computing and that each I/O operation takes 10 milliseconds to complete. Also assume that the context-switching overhead is 0.1 millisecond and that all processes are long running tasks. Describe the CPU utilization for a round-robin scheduler in the following two cases.

"Describe the CPU utilization" means show the sequence of CPU activities. Use 'I' to describe 1ms of execution of an I/O-process, 'C' to describe 1ms of execution of the CPU-bound process, 'o' for context-switching. The answer should be a sequence of C's, I's, and o's. Assume all processes are in the READY state initially and (arbitrarily) assume that I/O-bound processes a run before the CPU-bound process in the Round-Robin order. For each question give the percentage of time spent due to the context-switching overhead.

a. The time quantum is 1 millisecond

Answer: Since each I/O-bounded task issues an I/O operation every millisecond and it takes 10 milliseconds to complete, it definitely takes the overhead of context-switching every millisecond. The CPU-bounded task can't finish within one time quantum, and takes the overhead of 0.1 millisecond of context-switching every millisecond as well. The execution sequence of CPU is shown as the Gantt chart as below (each cell represents 1 millisecond of execution time, except for the 'o' cells, which represent 0.1 millisecond of context-switching):

```
--------------------------------------------------------------------------------------------
|   I1    | o |   I2    | o |   I3    | o |   I4    | o |   I5    | o |   I6    | o | (continue)
--------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------
|   I7    | o |   I8    | o |   I9    | o |   I10    | o |   C1    | o | (continue)
-------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------
|   I1    | o |   I2    | o |   I3    | o |   I4    | o |   I5    | o |   I6    | o | (continue)
--------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------
|   I7    | o |   I8    | o |   I9    | o |   I10    | o |   C1    | o | (continue)
-------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------
|   I1    | o |   I2    | o |   I3    | o |   I4    | o |   I5    | o |   I6    | o | (continue)
--------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------
|   I7    | o |   I8    | o |   I9    | o |   I10    | o |   C1    | o | (continue)
-------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------
|   I1    | o |   I2    | o |   I3    | o |   I4    | o |   I5    | o |   I6    | o | (continue)
--------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------
|   I7    | o |   I8    | o |   I9    | o |   I10    | o |   C1    | o | (continue)
-------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------
|   I1    | o |   I2    | o |   I3    | o |   I4    | o |   I5    | o |   I6    | o | (continue)
--------------------------------------------------------------------------------------------
--------------------------------------------------------------------
|   I7    | o |   I8    | o |   I9    | o |   I10    | o |   C1    | ···
--------------------------------------------------------------------
```

Therefore, regardless of the CPU-bounded or I/O-bounded tasks, the scheduler interrupts, and processes a 0.1 millisecond of context-switching time consumption every millisecond. The CPU utilization in this case is (just look at the first two rows of the scheduling because it's the same ratio in any larger scale):

$$\frac{10 \times 1 + 2 \times 1}{10 \times 1 + 2 \times 1 + 12 \times 0.1} = \frac{12}{13.2}$$

$$= 90.9\% \,.$$

b. The time quantum is 10 milliseconds

Answer: The behavior of the I/O-bounded tasks remains the same. They would issue an I/O operation every millisecond and takes a 0.1 millisecond context-switching overhead every millisecond. The CPU-bounded task, however, behaves differently. It would be able to finish the task within the time quantum, and takes only one time context-switching overhead. The execution sequence of CPU is shown as the Gantt chart as below (the small cells represent 1 millisecond of execution time, large cells represent 10 millisecond of execution time, and 'o' cells represent 0.1 millisecond of context-switching):

```
-------------------------------------------------------------------------------------
|   I₁   | o |   I₂   | o |   I₃   | o |   I₄   | o |   I₅   | o |   I₆   | o |   I₇   | o |
-------------------------------------------------------------------------------------
---------------------------------------------------------------------------
|   I₈   | o |   I₉   | o |   I₁₀   | o |            C₁            | o | (continue)
---------------------------------------------------------------------------
-------------------------------------------------------------------------------------
|   I₁   | o |   I₂   | o |   I₃   | o |   I₄   | o |   I₅   | o |   I₆   | o |   I₇   | o |
-------------------------------------------------------------------------------------
---------------------------------------------------------------------------
|   I₈   | o |   I₉   | o |   I₁₀   | o |            C₁            | o | (continue)
---------------------------------------------------------------------------
-------------------------------------------------------------------------------------
|   I₁   | o |   I₂   | o |   I₃   | o |   I₄   | o |   I₅   | o |   I₆   | o |   I₇   | o |
-------------------------------------------------------------------------------------
---------------------------------------------------------------------------
|   I₈   | o |   I₉   | o |   I₁₀   | o |            C₁            | o | (continue)
---------------------------------------------------------------------------
-------------------------------------------------------------------------------------
|   I₁   | o |   I₂   | o |   I₃   | o |   I₄   | o |   I₅   | o |   I₆   | o |   I₇   | o |
-------------------------------------------------------------------------------------
---------------------------------------------------------------------------
|   I₈   | o |   I₉   | o |   I₁₀   | o |            C₁            | o | (continue)
---------------------------------------------------------------------------
-------------------------------------------------------------------------------------
|   I₁   | o |   I₂   | o |   I₃   | o |   I₄   | o |   I₅   | o |   I₆   | o |   I₇   | o |
-------------------------------------------------------------------------------------
------------------------------------------------------------------
|   I₈   | o |   I₉   | o |   I₁₀   | o |            C₁            | ···
------------------------------------------------------------------
```

The CPU utilization in this case is roughly (just look at the first two rows of the scheduling because it's the same ratio in any larger scale):

$$\frac{10\times 1+1\times 10}{10\times 1+1\times 10+11\times 0.1}=\frac{20}{21.1}$$
$$=94.8\%\,.$$

Exercise 5.16: Consider a system implementing multilevel queue scheduling. What strategy can a computer user employ to maximize the amount of CPU time allocated to the user's process?

Answer: The multilevel feedback queue implements a special CPU scheduling. If a process doesn't use the entire quantum assigned to it, the OS assumes that this process is waiting for some I/O device, and should be kept in the high-priority queue so that is can get the CPU as soon as possible later. On the other hand, if the process does use its entire assigned time quantum, the OS thinks this is a time consuming process and moves it to the lower priority queue. This way, the system always favors the new processes and executes them first.

So, in order to maximize the amount of CPU time, the user process wants to stay in the high priority queue, and not be pushed to the lower priority queue when it doesn't finish within one time quantum. Pretending to be an I/O-bounded process is a strategy. According to the mechanism described in the first paragraph, if the process doesn't use up the time quantum, but lingering onto the CPU, the OS would assume it's an I/O-bounded process and keep it in the high priority queue.