CS372 Operating System
HW3

5.9 Why is it important for the scheduler to distinguish I/O-bound programs from CPU-bound programs?

Answer:

I/O-bound programs have the property of performing only a small amount of computation before performing I/O. Such programs typically do not use up their entire CPU quantum. CPU-bound programs, on the other hand, use their entire quantum without performing any blocking I/O operations. Consequently, one could make better use of the computer's resouces by giving higher priority to I/O-bound programs and allow them to execute ahead of the CPU-bound programs.

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

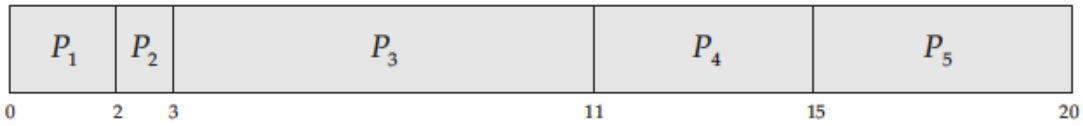
Process	Burst Time	Priority
P1	2	2
P2	1	1
P3	8	4
P4	4	2
P5	5	3

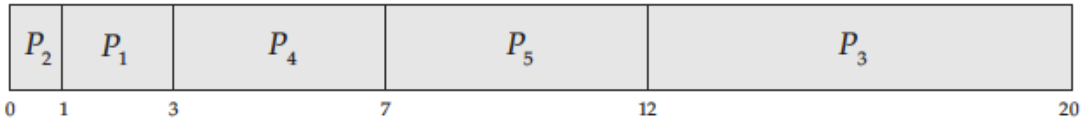
The processes are assumed to have arrived in the order **P**1, **P**2, **P**3, **P**4, **P**5, all at time 0.

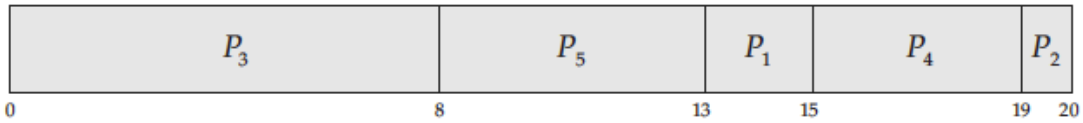
a. Draw four Gantt charts that illustrate the execution of these processes using the following scheduling algorithms: FCFS, SJF, nonpreemptive priority (a larger priority number implies a higher priority), and RR (quantum = 2).
b. What is the turnaround time of each process for each of the scheduling 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)?

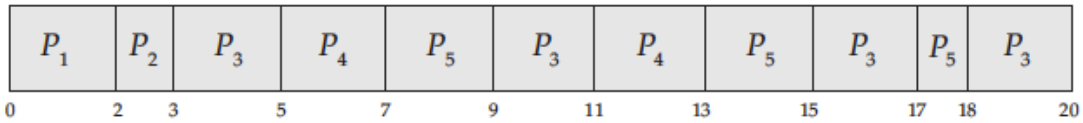
Answer:

a. The four Gantt charts are









b. Turnaround time

	FCFS	SJF	Priority	RR
P1	2	3	15	2
P2	3	1	20	3
P3	11	20	8	20
P4	15	7	19	13
P5	20	12	13	18

c. Waiting time (turnaround time minus burst time)

	FCFS	SJF	Priority	RR
P1	0	1	13	0
P2	2	0	19	2
P3	3	12	0	12
P4	11	3	15	9
P5	15	7	8	13

d. Shortest Job First

5.14 Consider a variant of the RR scheduling algorithm where the entries in the ready queue are pointers to the PCBs.

d. What would be the effect of putting two pointers to the same process in the ready queue?

a. What would be two major advantages and disadvantages of this scheme?

b. How would you modify the basic RR algorithm to achieve the same effect

without the duplicate pointers?

Answer:

a. In effect, that process will have increased its priority since by getting time more often it is receiving preferential treatment.
b. The advantage is that more important jobs could be given more time, in other words, higher priority in treatment. The consequence, of course, is that shorter jobs will suffer.
c. Allot a longer amount of time to processes deserving higher priority. In other words, have two or more quantums possible in the Round-Robin scheme.

6.9 The first known correct software solution to the critical-section problem for two processes was developed by Dekker. The two processes, **P**0 and **P**1, share the following variables:

```
boolean flag[2]; /* initially false */
int turn;
```

The structure of process **Pi** (i == 0 or 1) is shown in Figure 6.43; the other process is **Pj** (j == 1 or 0). Prove that the algorithm satisfies all three requirements for the critical-section problem.

Answer:

This algorithm satisfies the three conditions of mutual exclusion.
(1) Mutual exclusion is ensured through the use of the flag and turn variables. If both processes set their flag to true, only one will succeed, namely, the process whose turn it is. The waiting process can only enter its critical section when the other process updates the value of turn.
(2) Progress is provided, again through the flag and turn variables. This algorithm does not provide strict alternation. Rather, if a process wishes to access their critical section, it can set their flag variable to true and enter their critical section. It sets turn to the value of the other process only upon exiting its critical section. If this process wishes to enter its critical section again—before the other process—it repeats the process of entering its critical section and setting turn to the other process upon exiting.
(3) Bounded waiting is preserved through the use of the turn variable. Assume two processes wish to enter their respective critical sections. They both set their value of flag to true; however, only the thread whose turn it is can proceed; the other thread waits. If bounded waiting were not preserved, it would therefore be possible that the waiting process would have to wait indefinitely while the first process repeatedly entered—and exited—its critical section. However, Dekker's algorithm has a process set the value of turn to the other process, thereby ensuring that the other process will enter its critical section next.

6.11 What is the meaning of the term *busy waiting*? What other kinds of waiting are there in an operating system? Can busy waiting be avoided altogether? Explain your

answer.

Answer:

Busy waiting means that a process is waiting for a condition to be satisfied in a tight loop without relinquishing the processor. Alternatively, a process could wait by relinquishing the processor, and block on a condition and wait to be awakened at some appropriate time in the future. Busy waiting can be avoided but incurs the overhead associated with putting a process to sleep and having to wake it up when the appropriate program state is reached.

6.21 Write an algorithm for a bounded-buffer monitor in which the buffers (portions) are embedded within the monitor itself.

```
monitor bounded_buffer {
        int items[MAX ITEMS];
        int numItems = 0;
        condition full, empty;

        void produce(int v) {
                while (numItems == MAX ITEMS) full.wait();
                items[numItems++] = v;
                empty.signal();
        }

        int consume() {
                int retVal;
                while (numItems == 0) empty.wait();
                retVal = items[--numItems];
                full.signal();
                return retVal;
        }
}
```