Andre J Plath
Professor Yanwei Wu
CS365 Operating Systems and Networking
February 15, 2018

CS365 Operating System and Networking
Homework 3

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

I/O-bound programs can performing only a small amount of computation before performing I/O. These programs typically don't use up their complete CPU quantum. However, CPU-bound programs use their entire quantum without performing any blocking I/O operations by giving higher priority to I/O-bound programs and allowing them to execute ahead of the CPU-bound programs, makes better use computer's resources.

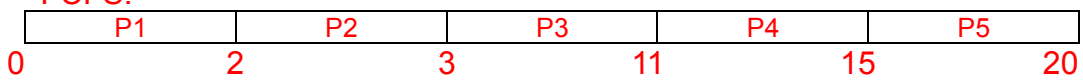
5.12 Consider the following set of processes, with the length of the CPU burst time given in milliseconds:

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

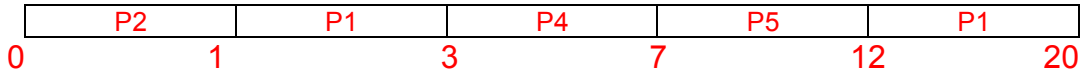
The processes are assumed to have arrived in the order **P**1, **P**2, **P**3, **P**4, **P**5, all at time 0.

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).

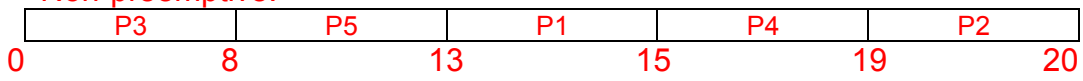
FCFS:



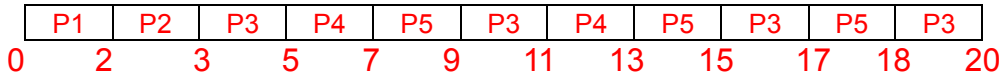
SJF:



Non-preemptive:



RR:



b. What is the turnaround time of each process for each of the scheduling

algorithms in part a?

FCFS	RR	SJF	Priority
P_1 2	2	3	15
P_2 3	3	1	20
P_3 11	20	20	8
P_4 15	13	7	19
P_5 20	18	12	13

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

FCFS	RR	SJF	Priority
P_1 0	0	1	13
P_2 2	2	0	19
P_3 3	12	12	0
P_4 11	9	3	15
P_5 15	13	7	8

d. Which of the algorithms results in the minimum average waiting time (over all processes)?
FCFS: 6.2
SJF: 4.2 = This is the fastest algorithm.
Preemptive: 11
RR: 7.2

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

a. What would be the effect of putting two pointers to the same process in the ready queue?
Process runs twice as many times.

a. What would be two major advantages and disadvantages of this scheme?
Pros:
1. Allows the user to prioritize more important processes.
2. Prevents starvation of lower priority processes.
3. Don't have to change the scheduling algorithm.
Cons:
1. Context switching now has a larger effect than it did before.
2. Removing processes from the running queue is now harder, would have to search through the whole list.

b. How would you modify the basic RR algorithm to achieve the same effect without the duplicate pointers?
Allow the quantum time each process gets to be changed on a per process basis.

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.

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. A waiting process can only enter its critical section when the other process updates the value of turn.
2. Progress is provided through the flag and turn variables. 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 if exiting its critical section. If this process wishes to enter its critical section again, before the other process, then it repeats the process of entering its critical section and setting turn to the other process when exiting.
3. Bounded waiting is also preserved through the use of the turn variable. 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.

Busy waiting is when a process is waiting for a condition to be met in a tight loop without surrendering the processor. A process could wait by giving up the processor and block on a condition and wait to be awoken in the future. Busy waiting can be avoided but at the cost of overhead, sleeping and waking.

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 Producer, Consumer;

void produce(int v)
 {
while (numItems == MAX_ITEMS) Producer.wait();
items[numItems++] = v;
 Consumer.signal();
}
```

```
int consume()
{
 int retVal;
while (numItems == 0) Consumer.wait();
count—
Producer.signal();
}
```