Operating Systems I

CS 474

Homework #2

Questions from Chapter 5

5.3 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. (6 pts)

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.

5.10 Explain why implementing synchronization primitives by disabling interrupts is not appropriate in a single-processor system if the synchronization primitives are to be used in user-level programs. (5 pts)

If a user-level program is given the ability to disable interrupts, then it can disable the timer interrupt and prevent context switching from taking place, thereby allowing it to use the processor without letting other processes to execute.

5.14 Describe how the compare and swap() instruction can be used to provide mutual exclusion that satisfies the bounded-waiting requirement. (15 pts)

```
do
{
        waiting [i] = TRUE;
        key = TRUE;
        while (waiting[i] && key)
                swap(&lock, &key);
        waiting [i] = FALSE;
        /*Critical Section*/
        j = (i+1) % n;
        while (j!=i && !waiting[i])
                j = (j+1) % n;
        if (j==i)
                lock = FALSE;
        else
                waiting [j] = FALSE;
        /*Remainder Section*/
        } while (TRUE);
```

5.17 Assume that a system has multiple processing cores. For each of the following scenarios, describe which is a better locking mechanism—a spinlock or a mutex lock where waiting processes sleep while waiting for the lock to become available: (6 pts)

• The lock is to be held for a short duration.
Spinlock – because it could be faster than a mutex which needs to suspend then awaken the waiting process
• The lock is to be held for a long duration.
Mutex – this allows other core to schedule another process while the locked one waits.
• A thread may be put to sleep while holding the lock.
Mutex - because a spinlock would be inefficient while waiting for the other process to wake.

Questions from Chapter 6

6.6 Suppose that a scheduling algorithm (at the level of short-term CPU scheduling) favors those processes that have used the least processor time in the recent past. Why will this algorithm favor I/O-bound programs and yet not permanently starve CPU-bound programs? (6 pts)

It will favor the I/O-bound programs because of the relatively short CPU burst request by them; however, the CPU-bound programs will not starve because the I/O-bound programs will relinquish the CPU relatively often to do their I/O.

6.11 Discuss how the following pairs of scheduling criteria conflict in certain settings. (9 pts)

- CPU utilization and response time
- **CPU utilization is increased if the overheads associated with context switching is minimized. The context switching overheads could be lowered by performing context switches infrequently. This could however result in increasing the response time for processes.**
- Average turnaround time and maximum waiting time
- **Average turnaround time is minimized by executing the shortest tasks first. Such a scheduling policy could however starve long-running tasks and thereby increase their waiting time.**
- I/O device utilization and CPU utilization
- **CPU utilization is maximized by running long-running CPU-bound tasks without performing context switches. I/O device utilization is maximized by scheduling I/O-bound jobs as soon as they become ready to run, thereby incurring the overheads of context switches.**

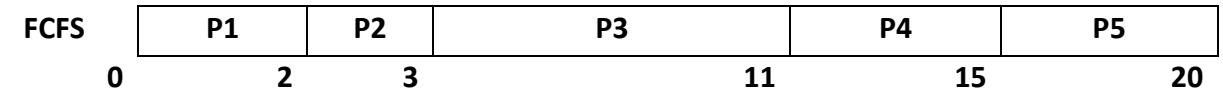
6.16 Consider the following set of processes, with the length of the CPU burst given in milliseconds: (12 pts)

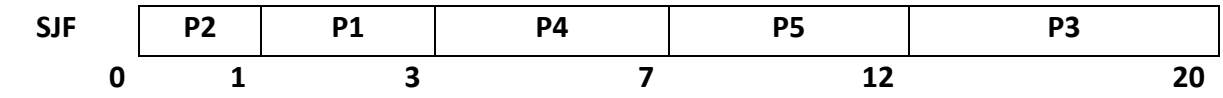
Thread	Burst	Priority
P_1	2	2
P_2	1	1
P_3	8	4
P_4	4	2
P_5	5	3

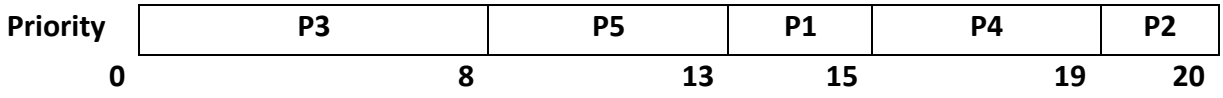
The processes are assumed to have arrived in the order P_1, P_2, P_3, P_4, P_5, all at time 0.

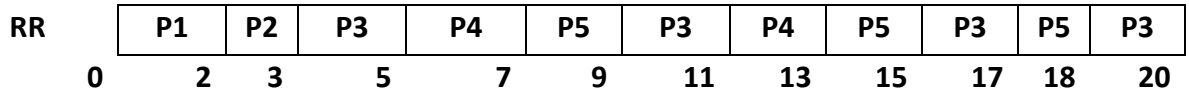
1) Draw four Gantt charts that illustrate the execution of these processes using the following scheduling algorithms: FCFS, SJF, nonpreemptive priority (a larger priority number implies a higher priority), and RR (quantum = 2).

   The Gantt charts are









2) What is the turnaround time of each process for each of the scheduling algorithms?

	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**

3) What is the waiting time of each process for each of these scheduling algorithms?

	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**

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

The SJF has the min average waiting time (4.6).

6.17 The following processes are being scheduled using a preemptive, round-robin scheduling algorithm. 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 Pidle). 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. (12 pts)

Thread	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

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

P1	P1	Pidle	P2	P3	P2	P3	P4	P4	P2	P3	Pidle	P5	P6	P5

0 10 20 25 35 45 55 60 70 75 80 90 100 105 115 120

2) What is the turnaround time for each process?

	Turnaround Time
P1	**20**
P2	**55**
P3	**60**
P4	**15**
P5	**20**
P6	**10**

3) What is the waiting time for each process?

	Waiting Time
P1	**0**
P2	**30**
P3	**35**
P4	**0**
P5	**10**
P6	**0**

4) What is the CPU utilization rate?

105/120 = 87.5%

Questions from Chapter 7

7.10 Is it possible to have a deadlock involving only one single-threaded process? Explain your answer. (5 pts)

It is not possible to have circular wait with only one process, thus failing a necessary condition for Circular wait. There is no second process to form a circle with the first one. So it is not possible to have a deadlock involving only one process.

7.16 In a real computer system, neither the resources available nor the demands of processes for resources are consistent over long periods (months). Resources break or are replaced, new processes come and go, and new resources are bought and added to the system. If deadlock is controlled by the banker's algorithm, which of the following changes can be made safely (without introducing the possibility of deadlock), and under what circumstances? (12 pts)

1) Increase Available (new resources added).

   This could safely be changed without any problems.

2) Decrease Available (resource permanently removed from system).

   This could have an effect on the system and introduce the possibility of deadlock as the safety of the system assumed there were a certain number of available resources.

3) Increase Max for one process (the process needs or wants more resources than allowed).

   Increase Max for one process (the process needs more resources than allowed, it may want more)—This could have an effect on the system and introduce the possibility of deadlock.

4) Decrease Max for one process (the process decides it does not need that many resources).

   This could safely be changed without any problems.

5) Increase the number of processes.

   This could be allowed assuming that resources were allocated to the new process(es) such that the system does not enter an unsafe state.

6) Decrease the number of processes.

   This could safely be changed without any problems.

7.23 Consider the following snapshot of a system: (12 pts)

	Allocation	*Max*	*Available*
	A B C D	*A B C D*	*A B C D*
P_0	2 0 0 1	4 2 1 2	3 3 2 1
P_1	3 1 2 1	5 2 5 2	
P_2	2 1 0 3	2 3 1 6	
P_3	1 3 1 2	1 4 2 4	
P_4	1 4 3 2	3 6 6 5	

Answer the following questions using the banker's algorithm:

1) Illustrate that the system is in a safe state by demonstrating an order in which the processes may complete.
2) If a request from process P1 arrives for (1, 1, 0, 0), can the request be granted immediately?
3) If a request from process P4 arrives for (0, 0, 2, 0), can the request be granted immediately?

1) **The system is in a safe state and the sequence < P0, P3, P4, P1, P2> satisfies the safety criteria (multiple sequences exist that satisfy the requirement).**
2) **The request from P1 for (1, 1, 0, 0) can be granted immediately.**
3) **The request from P4 for (0, 0, 2, 0) cannot be granted immediately.**