Benjamin Horn
Operating Systems HW 2
10/9/2017

Problem 1

Race conditions are caused when an outcome can be dependent on the order in which processes run. That is the order in which the threads or processes are called on. To prevent a race condition, you can synchronize processes running at the same time. For example, in a program where you want to multiply a value x by five then square it in another process, running those processes in that order is imperative to get the correct output. If it were to be squared then multiplied, depending on your value of x, your output could differ greatly. To avoid this race condition in updating our value x, the processes need to be mutually exclusive. Being mutually exclusive means a process that is using a variable that is shared (accessible by other processes), it is not used by another process until that one is done. When this race, or competition, occurs between two processes for one resource it is called the critical section or critical region. When one process is using/ in this critical section, no other process can enter into or conflict in this section. To avoid race conditions multiple processes cannot be inside the critical region (Mutual Exclusion). When no process is in the critical region, another cannot be blocked from entering the critical region. No process should wait forever to enter into its critical region, but it still must abide by the above rules. Lastly, assumptions cannot be made about speeds or the processes or the number of CPUs handling the load.

Problem 2

Process	Burst Time	Priority
P1	10	3
P2	1	1
P3	2	3
P4	1	4
P5	5	2

A)

Gantt Table 1 - FCFS

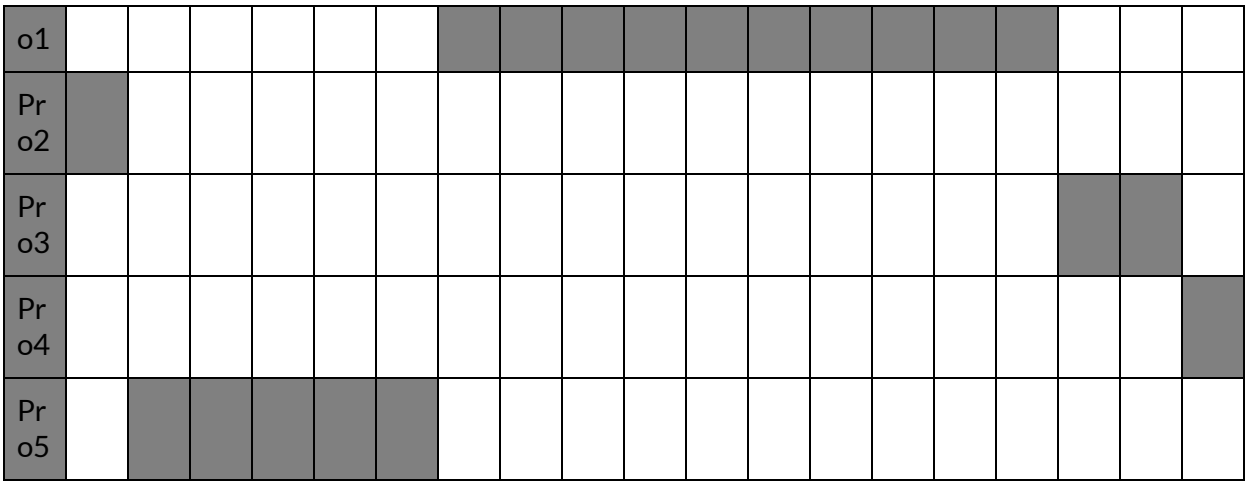
Process	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Pro1																			
Pro2																			
Pro3																			
Pro4																			
Pro5																			

Gantt Table 2 - SJF

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Pro1																			
Pro2																			
Pro3																			
Pro4																			
Pro5																			

Gantt Table 3 - Nonpreemptive priority

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Pr																			



Gantt Table 4 - Round Robin

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Pro1	■					■			■		■		■		■	■	■	■	■
Pro2		■																	
Pro3			■				■												
Pro4				■															
Pro5					■			■		■		■		■					

Turnaround times for each process of the scheduling algorithm.

	FCFS	SJF	NP	RR
Pro1	10	19	16	19
Pro2	11	1	1	2
Pro3	13	4	18	7
Pro4	14	2	19	4

Pro5	19	9	6	14

Waiting times for each process of the scheduling algorithm.

	FCFS	SJF	NP	RR
Pro1	0	9	6	9
Pro2	10	0	0	1
Pro3	11	2	16	5
Pro4	13	1	18	3
Pro5	14	4	1	9

SJF scheduling algorithm results in the minimal average wait time for the processes.

Problem 3

The scheduling algorithm, at the level of short-term CPU scheduling, will favor the i/o bound programs. This prioritization is due to the relatively short CPU burst requested by the i/o bound programs. CPU bound programs will not be starved out by this prioritization. The i/o bound programs will relinquish the CPU relatively often to do their i/o. When it is waiting for output, it gives up the time by using an interrupt to the CPU bound programs; allowing the CPU bound programs to work 'between' i/o bound programs.

Problem 4

Spinlock is a lock which makes the thread trying to acquire it wait. This wait is similar to a loop, where it will continually check if the lock has been made available. This spinlock will consume a CPU's resources while it retries over and over. In a uniprocessor system, a system which only has one central unit to execute and complete the given task, its resources will be eaten up by the spinlock. When you have a more than one CPU's available, a spinlock will not have as great of an effect.

There is a significant overhead caused by context switching and or rescheduling, which makes a spinlock useful and more efficient if the wait time is meager. However, if the spinlock has to wait for more extended amounts of time, it becomes inefficient, where a reschedule or context switch would be more beneficial for the process. This effect is amplified on a single processor system because processed are waiting for the thread to give up its lock and are just spinning waiting.

Problem 5

doemac:opsys benjamin.horn$ ./a.out
.... Calculating delay factor ......
.... End calculating delay factor
Reader 0 (pid = 14957) arrives
    Reader 0 gets results = **0000000000**

Writer 0 (pid = 14959) arrives, writing **0000000000** to buffer
Writer 0 finishes
Reader 1 (pid = 14960) arrives
    Reader 1 gets results = **0000000000**

Reader 2 (pid = 14961) arrives
    Reader 2 gets results = **0000000000**

Reader 3 (pid = 14962) arrives
    Reader 3 gets results = **0000000000**

Writer 1 (pid = 14963) arrives, writing **1111111111** to buffer
Writer 1 finishes
Reader 4 (pid = 14964) arrives
    Reader 4 gets results = **1111111111**

Writer 2 (pid = 14965) arrives, writing **2222222222** to buffer
Writer 2 finishes
Writer 3 (pid = 14966) arrives, writing **3333333333** to buffer
Writer 3 finishes
Reader 5 (pid = 14967) arrives
    Reader 5 gets results = **3333333333**

Reader 6 (pid = 14968) arrives
    Reader 6 gets results = **3333333333**

Reader 7 (pid = 14969) arrives
    Reader 7 gets results = **3333333333**

Writer 4 (pid = 14970) arrives, writing **4444444444** to buffer
Writer 4 finishes
Reader 8 (pid = 14971) arrives
    Reader 8 gets results = **4444444444**