CS 620: Homework 7

Carmen St. Jean

October 18, 2012

1. *(6) Calculate the mean response time (R) for the following scheduling policies using the workload given in Table 1. There are 3 Queues: qt1=1; qt2=2; qt3=3.*

Process	Arrival Time	CPU Burst
P1	0	3
P2	2	5
P3	3	4
P4	6	7
P5	9	3

Assume that context switch time is 0. If a job arrives at the same time that another job finishes its quantum, assume that the arriving job gets into the CPU queue ahead of the job that just finished its quantum.

(a) *MLF non-preemptive*

MLF	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	time
P1	E1	E2	E2																					3 - 0 = 3
P2			W1	E1	W2	E2	E2	W3	W3	W3	W3	W3	W3	W3	W3	E3	E3							17 - 2 = 15
P3				W1	E1	W2	W2	W2	E2	E2	W3	W3	W3	W3	W3	W3	W3	E3						18 - 3 = 15
P4							W1	E1	W2	W2	W2	E2	E2	W3	W3	W3	W3	W3	E3	E3	E3	E3		22 - 6 = 16
P5										W1	E1	W2	W2	E2	E2									15 - 9 = 6

$R_{MLF} = \frac{3+15+15+16+6}{5} = \frac{55}{5} = 11$ time units

(b) *MLF preemptive*

MLF pre	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	time
P1	E1	E2	W2	W2	E2																			5 - 0 = 5
P2			E1	W2	W2	E2	W2	W2	W2	W2	E2	E2	W3	W3	W3	W3	W3	E3						18 - 2 = 16
P3				E1	W2	W2	W2	E2	E2	W3	W3	W3	W3	W3	W3	W3	E3							17 - 3 = 14
P4							E1	W2	W2	W2	W2	W2	E2	E2	W3	W3	W3	W3	E3	E3	E3	E3		22 - 6 = 16
P5										E1	W2	W2	W2	W2	E2	E2								16 - 9 = 7

$R_{MLFpre} = \frac{3+16+14+16+7}{5} = \frac{58}{5} = 11.6$ time units

2. *(3) Consider the Feedback part of MLF: In class, I explained how jobs drop into lower priority queues. Design a computationally simple approach for jobs to climb back to an higher priority queue as they become I/O intensive.*

Whenever an I/O interrupt occurs before a job's quantum time is over, move the job to a higher priority queue rather than placing it back into the same queue.

3. *(4) Suppose you are given the job of developing the design of theMLF scheduling algorithm. You have to choose between the following: (1) a single linked list to represent all themultiple queues; or (2) multiple linked lists where each linked list represents a queue.*

(a) *Explain one advantage that the single linked list approach has over the multiple linked list approach.*

A single linked-list is easier to implement. It would also be simpler to change a job's priority because it would only require changing a value of the job's node in the list rather than move the node around to the appropriate position in the appropriate list.

(b) *Explain one advantage the the multiple linked list approach has over the single linked list approach.*

Multiple linked-lists would be easier to manage because you would have instant access to the top highest priority job in each list. You would also be able to quickly determine whether or not a specific list was empty.

4. *(9) The following is a variation of the critical section codes described in class. (Initially, flag[0] and flag[1] are both false.)*

```
THREAD 0
for (;;) {
    ...
    flag[0] = true;
    while (flag[1] == true);
    <critical section>
    flag[0] = false;
    <non-critical section>
    ...
}
THREAD 1
for (;;) {
    ...
    while (flag[0] == true);
    flag[1] = true;
    <critical section>
    flag[1] = false;
    <non-critical section>
    ...
}
```

(a) *Is mutual exclusion of the critical section guaranteed?*

No, both can easily enter the critical section at once.

(b) *Can Thread 0 starve out Thread 1?*

No, it's impossible T1 to start T0 because they are not mutually exclusive.

(c) *Can Thread 1 starve out Thread 0?*

No, it's impossible T0 to start T1 because they are not mutually exclusive.

(d) *Is deadlock possible?*

No, deadlock is not possible because either of them can enter the critical zone whenever it wants.

(e) *If a thread dies in its* <non−critical section>, *can it block the other thread?*

Yes.

5. *(3) Consider the question raised in class: if the* while *loop and the* turn = *commands can be interrupted, then would the code (discussed in class) execute correctly (i.e., mutual exclusion of critical section remains enforced)?*

No, the code may not execute correctly. Say that T1 is changing turn from 1 to 2, but is interrupted in the middle of this write. You could end up with the value of 0 in turn, which would make your while loop conditions impossible to satisfy...

This is how turn appears in binary before T1 attempts to write to it:

```
00000001
```

This is how turn should look after T1 changes it so it contains the value 2:

```
00000010
```

But the pipe below indicates where the interrupt occurs, so T1 does not manage to change the last bit of turn:

```
0000001|1
```

Meanwhile, T2 successfully changes the value of turn to 1:

```
00000001
```

But after the interrupt finishes, T1 attempts to pick up where it left off in modifying turn, so it changes that last bit to a zero. Thus turn will look like this:

```
00000000
```