CMSC 421 Homework 1 Answer Key

4-14-2006

Solution (1)

Consider a ready queue with four processes and their corresponding information as below:

Process	Arrival Time	Burst Time (ms)
P1	0	25
P2	0	14
P3	4	12
P4	5	18

Assume that each context switching takes 1 ms. Determine the average waiting time per process (including context switching times) for a round robin system with time quantum values of: (a) 4 ms and (b) 8 ms.

Solution (a)

P1 runs for 4 ms and P2 will wait for 5 ms (4 ms plus context switch). P3 is added at the end of the context switch, and thus waits 1 ms during the context switch, and P4 is subsequently added at the end of the context switch. Also, P1 waits 1 ms during the context switch. So, all processes have waited a total of 5 + 1 + 1 = 7 ms so far. Also note that the running queue is now full, and its current order is P2, P3, P4, P1, as we assume that a processes is added back onto the queue after the context switch completes (although this may very by implementation).

P2 will need 14/4 = 3 more full time quantums
P3 will need 12/4 = 3 more full time quantums
P4 will need 18/4 = 4 more full time quantums
P1 will need 21/4 = 5 more full time quantums

Note that the first process to finish will be P3, and therefore there will be 10 full time quantums before it finishes. 3*4 = 12 if every process runs, but it will finish before the 2 after it. During this time, each process besides the running process will wait 5 ms and the running process will wait for 1 ms, so for each full time quantum all processes will wait 16 ms total until process P3 ends at the 10th, in which case it does not wait the extra 1 ms.

$9 * 16 = 144$ ms
$150 + 15 + 6 = 165$ ms total, so far.

When P3 finishes, P4 is next to be run, and each processes Burst time above it was reduced by 12 and below it by 8 making the burst times for P1, P2, and P4 13, 2, and

10, respectively. At this point it is clear that P2 is the next to go, so P4 and P1 will run adding 11 ms each time to our total this time as P3 has completed, making our total 187.

When P2 completes the other two wait 3 ms (7 ms total), making our total 194. P1 and P4 now have burst times of 9 and 6 with P4 in the running queue. The rest is easy to compute as P1 waits 5 ms for P4 (6 ms total), P4 5 ms for P1 (6 ms total), and P1 3 ms for P4 (3 ms total, because P4 doesn't wait after it finishes). Adding 15 more to our total giving us a final value of:

209 So,

$$\frac{209}{4} = 52.25$$

Solution (b)

This problem is solved in the same way as **(a)**. To do this quickly, we present the following table, of which the top row details what Process is running and how many cycles it has left before it is run, and the rest of the table presents the wait time for each process.

	P1(25)	P2(14)	P3(12)	P4(18)	P1(17)	P2(6)	P3(4)	P4(10)	P1(9)	P4(2)
P1	1	9	9	9	1	7	5	9	1	3
P2	9	1	9	9	9	0	-	-	-	-
P3	5	9	1	9	9	7	0	-	-	-
P4	4	9	9	1	9	7	5	1	9	0

To get our answer, we simply total all the numbers in the columns and divide by the number of processes:

$$\frac{185}{4} = 46.25$$

Solution (2)

This question was clarified on March 30, 2006. Please note that t_0/τ_0 are not needed.

Consider a process with burst time (t_i) values of: $t_1 = 14$, $t_2 = 23$, $t_3 = 15$, $t_4 = 19$, $t_5 = 24$, $t_6 = 36$. Assume that $\tau_1 = t_1$. For alpha = 0.4 and alpha = 0.6, what is the value of τ_7?

$$\tau_{n+1} = \alpha t_n + (1-\alpha)\alpha t_{n-1} + (1-\alpha)^2\alpha t_{n-2} + ... + (1-\alpha)^{n+1} t_1$$

$\alpha = .4$:

$$\tau_7 = .4*36+(1-.4)*.4*24+(1-.4)^2*.4*19+(1-.4)^3*.4*15+(1-.4)^4*.4*23+(1-.4)^5*.4*14 = 25.819776$$

$\alpha = .6$:

$$\tau_7 = .6*36+(1-.6)*.6*24+(1-.6)^2*.6*19+(1-.6)^3*.6*15+(1-.6)^4*.6*23+(1-.6)^5*.6*14 = 30.199296$$

The above is acceptable. But, **an alternate (and actually accurate) solution for** $\alpha = 0.4$ is:

t_i	τ_i
14	14
23	$\alpha * 14 + (1-\alpha) * 14 = 14$
15	$\alpha * 23 + (1-\alpha) * 14 = 17.6$
19	$\alpha * 15 + (1-\alpha) * 17.6 = 16.56$
24	$\alpha * 19 + (1-\alpha) * 16.56 = 17.536$
36	$\alpha * 24 + (1-\alpha) * 17.536 = 20.1216$
-	$\alpha * 36 + (1-\alpha) * 20.1216 = 26.47296$

In a similar manner, repeat for $\alpha = 0.6$.

Solution (3)

Problem 5.6

Consider a variant of the RR scheduling algorithm in which the entires in the ready queue are pointers to the PCBs.

Solution (a)

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

The process would be run twice as many times.

Solution (b)

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

- Advantages
  1. It would allow the user to prioritize more important processes.
  2. It prevents starvation of lower priority processes.
  3. We do not have to change the scheduling algorithm.
- Disadvantage
  1. Context switching will now have a larger effect than it did before.
  2. Removing processes from the running queue is now significantly harder, as you would have to search through the whole list.

Solution (c)

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.

Solution (4)

Problem 5.9

Consider a preemptive priority scheduling algorithm based on dynamically changing priorities. Larger priority numbers imply higher priority. When a

process is waiting for the CPU (in the ready queue, but not running), its priority changes at a rate α; when it is running, its priority changes at a rate β. All processes are given a priority of 0 when they enter the ready queue. The parameters α and β can be set to give many different scheduling algorithms.

Solution (a)

What is the algorithm that results from $\beta > \alpha > 0$?

Since β is a rate and it is greater than that of α, processes that are running will continually receive a much higher priority. At the same time, the priority of processes waiting to be run will increase, however, this increase will not be as fast as the process running on the CPU. This will result in a first-come first-serve (FCFS) type of situation where when no process is running the one that has been waiting the longest will be ran, and its priority will continuously increase faster than the others until it stops.

Solution (b)

What is the algorithm that results from $\alpha < \beta < 0$?

New processes in this scheme will always be ran immediately, and older processes will sit in the ready queue until the most recent process finishes. This is essentially *Last In First Out*.

Solution (5)

Problem 7.2

Consider the deadlock situation that could occur in the dining philosophers problem when the philosophers obtain the chopsticks one at a time. Discuss how the four necessary conditions for deadlock indeed hold in this setting. Discuss how deadlocks could be avoided by eliminating any one of the four conditions

- **Mutual Exclusion**: When a philosopher picks up one chopstick, it cannot be shared with others. If they could be shared this situation would not exist and would prevent any deadlocks from occurring.
- **Hold and Wait**: When the philosopher tries to pick up a chopstick, he only picks up one at a time. If he could pick up both chopsticks at one time then a deadlock condition could not exist.
- **No preemption**: Once a philosopher picks up a chopstick, it cannot be taken away from her. If it could, then a deadlock condition could not exist.
- **Circular Wait**: Because all of the philosophers are sitting in a round table and each philosopher has access to the chopsticks next to them, if a philosopher picks up one chopstick he will affect the philosopher sitting next to him. and the philosopher on that side can also affect the philosopher sitting next to her in the same manner. This holds true all the way around the table. If one of the philosophers in the table could pick up a chopstick that another philosopher never needed, a deadlock condition would not exist.

Solution (6)

Problem 7.5

(a) Can be done safely

(b) Cannot be done safely

(c) Cannot be done safely

(d) Can be done safely

(e) Can be done safely, if resources allocated to new processes do not result in unsafe state.

(f) Can be done safely

Solution (7)

Problem 7.11

Consider the following snapshot of a system:

	Allocation	*Max*	*Available*
	A B C D	A B C D	A B C D
P_0	0 0 1 2	0 0 1 2	1 5 2 0
P_1	1 0 0 0	1 7 5 0	
P_2	1 3 5 4	2 3 5 6	
P_3	0 6 3 2	0 6 5 2	
P_4	0 0 1 4	0 6 5 6	

Answer the following questions using the banker's algorithm:

Solution (a)

What is the content of the matrix *Need*?

	Need
	A B C D
P_0	0 0 0 0
P_1	0 7 5 0
P_2	1 0 0 2
P_3	0 0 2 0
P_4	0 6 4 2

Solution (b)

Is the system in a safe state?

Yes, there exist several sequences that satisfy safety requirements (e.g. P_0, P_2, P_1, P_3, P_4).

Solution (c)

If a request from process P_1 arrives for (0, 4, 2, 0), can the request be granted immediately?

Pretend that the allocation can be made since the Available matrix is (1, 5, 2, 0), and it will now change to (1, 1, 0, 0). The next step is to find the safe sequence of processes. Alloc for P_1 becomes (1,4,2,0) and Need for P_1 becomes (0, 3, 3, 0).

One possible safe sequence is: P_0, P_2, P_3, P_1, P_4.

Solution (8)

A single-lane bridge connects the two Vermont villages of North Tunbridge and South Tunbridge. Farmers in the two villages use this bridge to deliver their produce to the neighboring town. The bridge can become deadlocked if both a northbound and a southbound farmer get on the bridge at the same time (Vermont farmers are stubborn and are unable to back up.) Using semaphores, design an algorithm that prevents deadlock. Initially, do not be concerned about starvation (the situation in which northbound farmers prevent southbound farmers from using the bridge, or vice versa).

A mutex semaphore is all that is needed to solve this problem. When a driver approaches the bridge they will reserve (request) the semaphore. If granted, they will cross the bridge, once they have crossed the bridge they will signal (release) the semaphore. If not, they will wait until they have received the semaphore.