TTk4145

Real-time Programming

Exercise 9

Spring 2017

by Dag Erik Homdrum Løvgren (delovgre)

1.

1.1. It enables us to complete tasks that are more important than others first. It enables us to prioritize tasks that have deadlines (or shorter deadlines) before others, which is important in real-time systems.

1.2. It must be able to meet deadlines for tasks. This is the main difference. In a real-time system some tasks have a hard deadline, eg. an automatic brake-system in a car needs to be able to signal the brake to brake within a certain timeframe from registering an incoming crash. These deadlines must be met for the system som do its job, and we need to be able to analyse the system and reason about guarantees for these deadlines.

2.

2.1.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
a					E							Q	V	E	
b			E	V		V	E	E	E						
c	E	Q								Q	Q				E

2.2.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
a					E			Q		V	E				
b			E	V					V			E	E	E	
c	E	Q				Q	Q								E

3.

3.1. Priority inversion is a problem in scheduling where a higher priority task is preempted by a lower priority task, *indirectly.* I say indirectly because such a thing would not be possible directly (in a reasonable system), but it can happen because of resource locks. Let's say a low priority task, L, aquires shared resource, R, before a high priority task, H. H then preempts L, and reaches a section where it requires R. The execution now goes to L, effectively inversing the priority of the two tasks.

Unbounded priority inversion occurs if a new medium priority task, M, preempts L, which is logical because it has higher priority, during L's execution of its critical region with R. This leads to H waiting on both M and L, and M does not even share a resource with H. Any number of M tasks can preempt L, hence the name unbounded.

3.2. Not in and of itself. This depends on the rest of the implementation of the system. Deadlocks could still occur because of nested resource locks.

You could, for an example, allow each task to own only one shared resource at a time. In combination with priority inheritance, this would avoid deadlocks.

4.

4.1. Assumptions/Conditions:

- We have a fixed set of tasks.
- All tasks are periodic and we know the periods.
- All tasks are independent of eachother.
- All deadlines are equal to the task's period.
- All tasks have a fixed worst-case execution time.
- We ignore all context-switching costs.
- We have no internal supension points, eg. delay or I/O.
- All tasks execute on a single processor.

Well. They're not very realistic at all. I would be positively blown away if I ever enountered a real system which met these conditions. I can't actually see that any one of these conditions is very realistic in a real life system.

4.2. We here have three tasks. This gives us the utilization bound:

$$n\left(2^{\frac{1}{n}}-1\right)=3\left(2^{\frac{1}{3}}-1\right)\approx 0.7798$$

We have to get a utilization below or equal to this to pass the test.

We have:

$$n=3, \qquad C=\{15,10,5\}, \qquad T=\{50,30,20\}$$

Which gives us the value:

$$U=\sum_{i=1}^{3}\frac{C_i}{T_i}=\frac{15}{50}+\frac{10}{30}+\frac{5}{20}=\frac{53}{60}\approx 0.8833$$

The value is above our bound, and our set does not pass the utilization test. It is therefore not schedulable (according to this test).

4.3. We have established that (according to rate monotonic priorities):

$$p(c) > p(b) > p(a)$$

We therefore calculate c before b before a (c has no higher priority tasks):

$$w_c^0 = 5$$
$$w_c^1 = 5$$
$$\rightarrow RT_c = 5$$

$$w_b^0 = 10$$
$$w_b^1 = 10 + \sum_{j \in \{c\}} \left\lceil \frac{w_b^0}{T_j} \right\rceil C_j = 10 + \left\lceil \frac{10}{20} \right\rceil 5 = 15$$
$$w_b^2 = 10 + \sum_{j \in \{c\}} \left\lceil \frac{w_b^1}{T_j} \right\rceil C_j = 10 + \left\lceil \frac{20}{20} \right\rceil 5 = 15$$
$$\rightarrow RT_b = 15$$

$$w_a^0 = 15$$
$$w_a^1 = 15 + \sum_{j \in \{b,c\}} \left\lceil \frac{w_a^0}{T_j} \right\rceil C_j = 15 + \left\lceil \frac{15}{30} \right\rceil 10 + \left\lceil \frac{15}{20} \right\rceil 5 = 30$$
$$w_a^2 = 15 + \sum_{j \in \{b,c\}} \left\lceil \frac{w_a^1}{T_j} \right\rceil C_j = 15 + \left\lceil \frac{30}{30} \right\rceil 10 + \left\lceil \frac{30}{20} \right\rceil 5 = 35$$
$$w_a^3 = 15 + \sum_{j \in \{b,c\}} \left\lceil \frac{w_a^2}{T_j} \right\rceil C_j = 15 + \left\lceil \frac{35}{30} \right\rceil 10 + \left\lceil \frac{35}{20} \right\rceil 5 = 45$$
$$w_a^4 = 15 + \sum_{j \in \{b,c\}} \left\lceil \frac{w_a^3}{T_j} \right\rceil C_j = 15 + \left\lceil \frac{45}{30} \right\rceil 10 + \left\lceil \frac{45}{20} \right\rceil 5 = 50$$
$$w_a^5 = 15 + \sum_{j \in \{b,c\}} \left\lceil \frac{w_a^4}{T_j} \right\rceil C_j = 15 + \left\lceil \frac{50}{30} \right\rceil 10 + \left\lceil \frac{50}{20} \right\rceil 5 = 50$$
$$\rightarrow RT_a = 50$$

We can see that all response times for all tasks are less than or equal to their period. Therefore, all deadlines are met, and the set is schedulable.

We did get a different result from the utilization test, which said the tests would not be schedulable. We say that passing the utilization test is sufficient, but not necessary, and that the response-time analysis is both. This is in agreement with our result.