

**RDCS423 Tutorial Problems & Solutions #1 - Real-Time Scheduling Theory**

1. Consider the case of three periodic tasks:

- Task  $t_1$ :  $C_1 = 20$  ms;  $T_1 = 100$  ms
- Task  $t_2$ :  $C_2 = 40$  ms;  $T_2 = 150$  ms
- Task  $t_3$ :  $C_3 = 100$  ms;  $T_3 = 350$  ms

Apply the Utilization Bound Theorem to determine if these tasks are schedulable using a rate monotonic scheduling strategy. Suppose the computation time for task 1 doubles to 40 msec, now determine if the tasks are schedulable, and then apply the less conservative Completion Time Theorem.

2. Suppose we have four tasks: two periodic, one aperiodic, and one interrupt driven aperiodic. The non-interrupt driven tasks require access to a shared data store, and we wish to give the interrupt-drive task the highest priority:

- periodic task  $t_1$ :  $C_1 = 30$  ms,  $T_1 = 100$  ms
- aperiodic task  $t_2$ :  $C_2 = 30$  ms,  $T_2 = 150$  ms
- interrupt driven aperiodic task  $t_a$ :  $C_a = 10$  ms,  $T_a = 200$  ms
- periodic task  $t_3$ :  $C_3 = 30$  ms,  $T_3 = 300$  ms

The context switch time is included in the indicated CPU times. Use the Generalized Utilization Bound Theorem to determine if this task set is schedulable.

3. Given two tasks  $T_1$  and  $T_2$  with two shared data structures protected with binary semaphores  $S_1$  and  $S_2$ , show how the *priority ceiling protocol* prevents mutual deadlock and guarantees that a high-priority task will be blocked by at most one critical section of any lower priority task.

**Solutions:**

1. Compute the utilizations for each task:

- Task  $t_1$ :  $C_1 = 20$  ms;  $T_1 = 100$  ms  $\rightarrow U_1 = 0.2$
- Task  $t_2$ :  $C_2 = 40$  ms;  $T_2 = 150$  ms  $\rightarrow U_2 = 0.267$
- Task  $t_3$ :  $C_3 = 100$  ms;  $T_3 = 350$  ms  $\rightarrow U_3 = 0.286$

i.e.  $U_{total} = 0.753$  Assume that the context switch overhead is included in the CPU times. The upper bound from the Utilization Bound Theorem is:

$$U(3) = 3(2^{1/3} - 1) = 0.78$$

which is greater than the total utilization of these tasks  $\rightarrow$  all three tasks can meet their deadlines.

Given that  $t_1$ 's performance changes to:

$$\text{Task } t_1: C_1 = 40 \text{ ms; } T_1 = 100 \text{ ms } \rightarrow U_1 = 0.4$$

Now,  $U_{total} = 0.953$  which is greater than the bound  $\rightarrow$  the tasks fail to meet their deadlines. Also the first two tasks can be checked in the same way, e.g.  $U_{total} = 0.667$  and the upper bound becomes:

$$U(2) = 2(2^{1/2} - 1) = 0.828$$

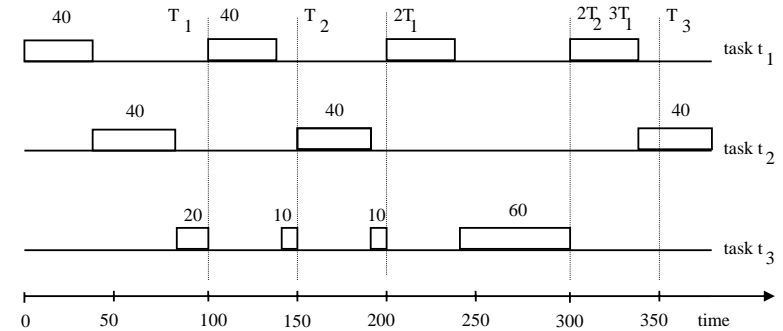
which is greater than the total utilization of these tasks  $\rightarrow$  at least the first two tasks can meet their deadlines.

The basis for applying the Completion Time Theorem is that provided each task completes execution before its first period (i.e. meets its first deadline) when all tasks are started at the same time, then the deadlines will be met for any task start times.

The worst-case scenario is with all three tasks ready to execute at the same time. Using rate monotonic scheduling  $t_1$  executes first, followed by  $t_2$  then  $t_3$ .

Note that task  $t_i$  will execute once for a CPU time of  $C_i$  during a period  $T_i$  and higher priority tasks will execute more often and may pre-empt task  $t_i$ . Thus it is necessary to consider the CPU time used by all higher priority tasks.

The ends of the first periods of each of the tasks,  $T_i$ , are referred to as *scheduling points*.



Look at the appropriate scheduling points for task 3, i.e. the ends of the periods of all higher priority tasks which have times less than or equal to task 3's period.

The completion time theorem checks that all tasks have completed their execution at any of the scheduling points (SP). For example, to check if all tasks met their deadlines at task 3's SP, i.e. 350 msec, we potentially have 4 executions of task 1, 3 executions of task 2 and 1 execution of task 3  $\rightarrow 4C_1 + 3C_2 + C_3 \leq T_3$ ? ( $160 + 120 + 100 > 350$ )

This doesn't mean the task set can't meet its deadlines - the theorem requires that all SP's be checked, i.e. if all tasks can meet their deadlines for one scheduling point, then the task set is schedulable. So looking at each scheduling point:

- $T_1$ :  $C_1 + C_2 + C_3 \leq T_1$ ? ( $40 + 40 + 100 > 100$ )
- $T_2$ :  $2C_1 + C_2 + C_3 \leq T_2$ ? ( $80 + 40 + 100 > 150$ )
- $2T_1$ :  $2C_1 + 2C_2 + C_3 \leq 2T_1$ ? ( $80 + 80 + 100 > 200$ )
- $3T_1$ :  $3C_1 + 2C_2 + C_3 \leq 3T_1$ ? ( $120 + 80 + 100 = 300$ )
- $2T_2$ :  $3C_1 + 2C_2 + C_3 \leq 2T_2$ ? ( $120 + 80 + 100 = 300$ )
- $T_3$ :  $4C_1 + 3C_2 + C_3 \leq T_3$ ? ( $160 + 120 + 100 > 350$ )

Thus the condition for SP  $3T_1$  is met, i.e. after 300 msec, task 1 will have run 3 times, task 2 will have run 2 times and task 3 once  $\rightarrow$  the required computation fits in this SP.

Note that this indicates that we could not add any higher priority tasks than task 3, otherwise it would miss its deadline. But tasks of lower priority than task 3 could be added if they have a sufficiently long period.

Applying the mathematical expression for the Completion Time Theorem where  $C_j$  is the execution time, and  $T_j$  is the period, of task  $t_j$ :

$$\forall i, 1 \leq i \leq n, \forall (k, p) \in R_i, \sum_{j=1}^i C_j \left\lceil \frac{pT_k}{T_j} \right\rceil \leq pT_k$$

where  $R_i = \{(k, p) \mid 1 \leq k \leq i, p = 1, \dots, \lfloor T_i / T_k \rfloor\}$  and at least one of the inequalities must be met for each  $i$ .

As an example, we must check task  $t_3$  against  $t_1$ : i.e.  $i = 3$  and  $k = 1$

$$\rightarrow p = 1, \dots, \lfloor T_i / T_k \rfloor = 1, \dots, \lfloor 350 / 100 \rfloor = 1, 2, 3.$$

$$\text{i.e. } R_i = (k, p) = (1, 1), (1, 2), (1, 3)$$

$$\begin{aligned} \text{e.g for } (k, p) = (1, 3) &\Rightarrow C_1 \left\lceil \frac{(3)T_1}{T_1} \right\rceil + C_2 \left\lceil \frac{(3)T_1}{T_2} \right\rceil + C_3 \left\lceil \frac{(3)T_1}{T_3} \right\rceil \leq (3)T_1 \\ &\Rightarrow C_1 \left\lceil \frac{(3)100}{100} \right\rceil + C_2 \left\lceil \frac{(3)100}{150} \right\rceil + C_3 \left\lceil \frac{(3)100}{350} \right\rceil \leq 3T_1 \\ 3C_1 + 2C_2 + C_3 &\leq 3T_1 \Rightarrow 120 + 80 + 100 = 300 \end{aligned}$$

and as at least one inequality is met, so the task set is schedulable (strictly, by the theorem, we would also have to check task 2 against task 1, and task 1 against itself).

2. First, determine the utilizations:

$$\text{periodic task } t_1: C_1 = 30 \text{ ms}, T_1 = 100 \text{ ms} \rightarrow U_1 = 0.3$$

$$\text{aperiodic task } t_2: C_2 = 30 \text{ ms}, T_2 = 150 \text{ ms} \rightarrow U_2 = 0.2$$

$$\text{interrupt driven aperiodic task } t_a: C_a = 10 \text{ ms}, T_a = 200 \text{ ms} \rightarrow U_a = 0.05$$

$$\text{periodic task } t_3: C_3 = 30 \text{ ms}, T_3 = 300 \text{ ms} \rightarrow U_3 = 0.1$$

Using rate monotonic priority assignment, the priorities would be  $t_1, t_2, t_a, t_3$ . Because a fast response is required to interrupts the priority of  $t_a$  is raised to be the highest.

The overall CPU utilization is 0.65 which is less than the utilization bound  $U(4) = 4(2^{1/4} - 1) = 0.76$ . Because of the non-rate monotonic priority assignment it is necessary to consider each task individually:

- A. Consider task  $t_a$  - highest priority with  $U_a = 0.05 \rightarrow$  no trouble meeting its deadline.
- B. Consider task  $t_1$  - apply the Generalized Utilization Bound Theorem:
  - a) Pre-emption by high-priority tasks with periods less than  $T_1$  (there are none here).
  - b) Execution utilization for task  $t_1$  is  $U_1 = 0.3$ .
  - c) Pre-emption by high-priority tasks with longer periods. Task  $t_a$  falls into this category  $\rightarrow$  utilization in the period of the task is  $C_a/T_1 = 10 \text{ ms}/100 \text{ ms} = 0.1$ .

- d) Blocking time by lower priority tasks. Both  $t_2$  and  $t_3$  can potentially block  $t_1 \rightarrow$  assuming the priority ceiling algorithm is being used, at most only one task can block  $t_1$ , so take the worst-case of  $t_3$  (since it has the longer execution time), i.e. blocking utilization during the period of the task is  $B_3/T_1 = 30 \text{ ms}/100 \text{ ms} = 0.3$ .

For task  $t_1$ , the worst case utilization =  $0.3 + 0.1 + 0.3 = 0.7$  which is less than the worst case utilization bound =  $0.76 \rightarrow$  task  $t_1$  will meet it's deadline.

- C. Consider task  $t_2$  - apply the Generalized Utilization Bound Theorem:
  - a) Pre-emption by high-priority tasks with periods less than  $T_2$ . Task  $t_1$  has a period less than  $T_2$ , so its pre-emption utilization during the period is  $U_1 = 0.3$ .
  - b) Execution utilization  $U_2 = 0.2$ .
  - c) Pre-emption by high-priority tasks with longer periods. Task  $t_a$  falls into this category  $\rightarrow$  utilization in the period of the task is  $C_a/T_2 = 10 \text{ ms}/150 \text{ ms} = 0.07$ .
  - d) Blocking time by lower priority tasks. Task  $t_3$  can potentially block  $t_2 \rightarrow$  again assuming the priority ceiling algorithm is being used, at most only one task can block  $t_2$ , so take the worst-case of  $t_3$ , i.e. blocking utilization during the period of the task is  $B_3/T_2 = 30 \text{ ms}/150 \text{ ms} = 0.2$ .

For task  $t_2$ , the worst case utilization =  $0.3 + 0.2 + 0.07 + 0.2 = 0.77$  which is greater than the worst case utilization bound =  $0.76 \rightarrow$  task  $t_2$  will just miss its deadline.

- D. Consider task  $t_3$  - apply the Generalized Utilization Bound Theorem:
  - e) Pre-emption by high-priority tasks with periods less than  $T_3$ . Tasks  $t_1, t_2$  and  $t_a$  have periods less than  $T_3$ , so its pre-emption utilization during the period is  $U_1 + U_2 + U_a = 0.3 + 0.2 + 0.05 = 0.55$ .
  - f) Execution utilization  $U_3 = 0.1$ .
  - g) Pre-emption by high-priority tasks with longer periods. There are no tasks in this category.
  - h) Blocking time by lower priority tasks. There are no lower priority tasks.

For task  $t_3$ , the worst case utilization =  $0.55 + 0.1 = 0.65$  which is less than the worst case utilization bound =  $0.76 \rightarrow$  task  $t_3$  will meet its deadline.

Thus all four tasks will not meet their deadlines (although task 2 is close to meeting its deadline). As usually the worst case upper bound is taken to be 0.69 to provide a reasonable safety-margin  $\rightarrow$  requires rescheduling of tasks.

- 3. Let  $T_1$  attempt to lock the semaphores in the order  $S_1$  then  $S_2$  and  $T_2$  attempt to lock in the reverse order. Also let  $T_1$  have higher priority than  $T_2$ . As both tasks use the semaphores, the priority ceilings of  $S_1$  and  $S_2$  must be set to that of task  $T_1$  or higher.

Let  $T_2$  start by acquiring  $S_2$  and when  $T_1$  is executed it will pre-empt  $T_2$ .  $T_1$  attempts to lock  $S_1$  but because its priority is not higher than the priority ceiling of already locked semaphore  $S_2$  it is suspended. Task  $T_2$  has its priority raised to that of the semaphore it has acquired ( $S_2$ ) and it now continues execution and acquires  $S_1$ .

Only one lower priority task can block  $T_1$  because when the semaphore that the lower priority task has (in this case  $T_2$ ) is returned,  $T_1$  continues execution with that semaphore (since it was suspended at that point and is the highest priority task waiting on that semaphore).