UPPSALA
UNIVERSITET

Real Time System

Lab: Response Time Analysis using FpsCalc

Group 02

Sagar Shubham Arijeet Laga Kedar Joshi
Aakash Goel

October 11, 2018

Contents

List of Figures

1 Quesion 1

Task	C_i	T_i	D_i
τ_1	2 ms	10 ms	10 ms
τ_2	4 ms	15 ms	15 ms
τ_3	10ms	35 ms	35 ms

Question 1.1

What is the priority ordering for the tasks using the RM priority ordering?

Rate Monotonic Priority Ordering assigns Priorities to a task of a task set inversely proportional to its period. Which is to say, smaller the period of a task, higher its priority.

Thus, for the given task set, the RM priority order is:

$$\tau_1 \prec \tau_2 \prec \tau_3$$

Question 1.2

Will all tasks complete before their deadlines according to the schedulability formula in Equation 2 on page 3? Draw a critical instant schedule for the given tasks (as in Figure 2, but have all tasks simultaneously released at time 0). What is the system utilization bound?

As per formula for utilization,

$$U = \sum_{i=1}^{n} \frac{C_i}{T_i}$$
$$U = \frac{2}{10} + \frac{4}{15} + \frac{10}{35}$$
$$U = 0.7523810$$

On the other hand, the upper bound for Utilization is given by

$$U_{max,RM} = n(2^{1/n} - 1)$$
$$U_{max,RM} = 3 \times (2^{1/3} - 1)$$
$$U_{max,RM} = 0.7797632$$

As $U \leq U_{max,RM}$, all the tasks will complete before their deadlines(As per Liu and Layland limited model).

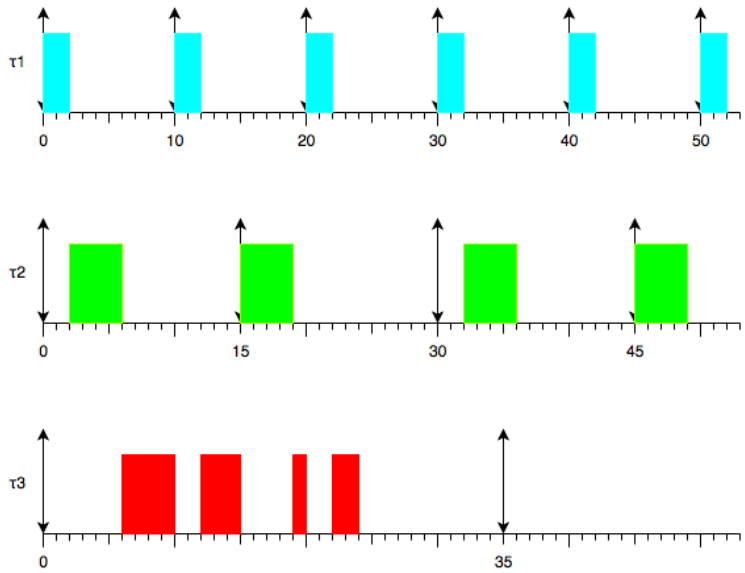


Figure 1: Schedule for (τ_1, τ_2, τ_3)

Question 1.3

Assume that we want to increase the computation time for task τ_1 to be C1 = 4. Will all task complete before their deadlines? Draw a critical instant schedule for the given tasks. What is the system utilization bound?

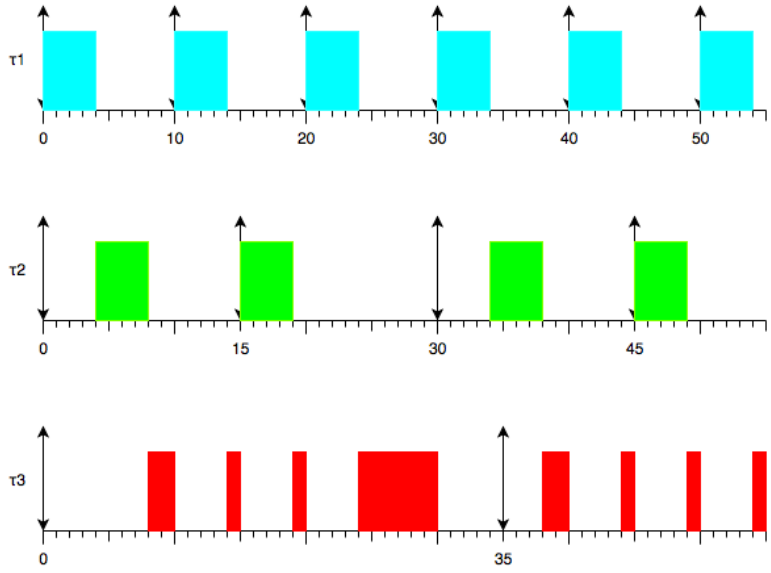


Figure 2: Schedule for (τ_1, τ_2, τ_3). Even when $U > U_{max,RM}$, deadline not missed.

Again, as per formula for utilization,

$$U = \sum_{i=1}^{n} \frac{C_i}{T_i}$$
$$U = \frac{4}{10} + \frac{4}{15} + \frac{10}{35}$$
$$U = 0.9523810$$

On the other hand, the upper bound for Utilization is still same at $U_{max,RM} = 0.7797632$.

As $U_{max,RM} \leq U \leq 1$, we cannot confirm if the tasks will miss their deadline or not (As $[U \leq U_{max,RM}]$ is a sufficient condition, but not a necessary condition). For example, we can see that from Figure 2, neither of the tasks miss their deadlines within the first $35ms$. To check if this scheduling actually works, we need to extrapolate the above figure up to the hyperperiod of the T, which is $L.C.M(10, 15, 35) = 210ms$, and see if any deadline miss occurs.

Question 1.4

Assume that we want to increase the computation time for task τ_1 to be C1 = 5. Will all task complete before their deadlines? Draw a critical instant schedule for the given tasks. What is the system utilization bound?

The Utilization now is

$$U = \sum_{i=1}^{n} \frac{C_i}{T_i}$$
$$U = \frac{5}{10} + \frac{4}{15} + \frac{10}{35}$$
$$U = 1.0523810$$

As $U > 1.0$, the tasks will not be schedulable, and will miss their deadlines if attempted to be scheduled by Rate Monotonic Scheduling Algorithm on a uni-core, pre-emptive processor. One such instance is shown in Figure 3 .

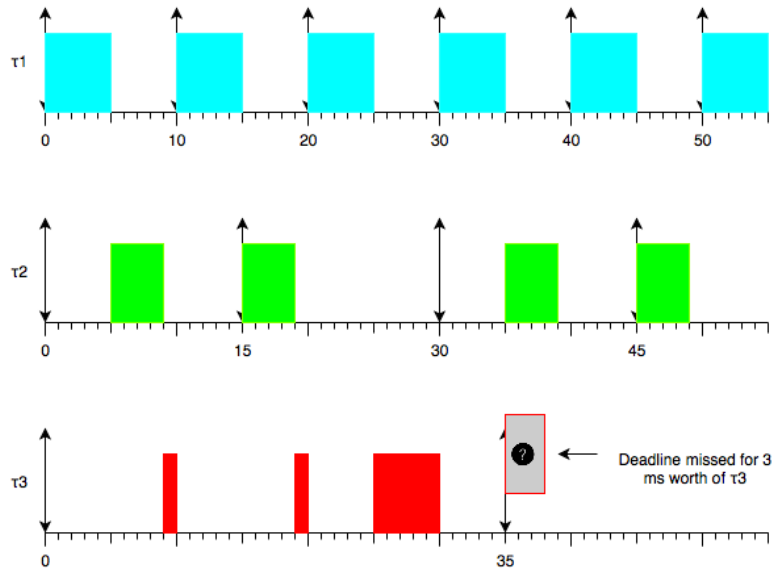


Figure 3: Schedule for (τ_1, τ_2, τ_3), Deadline Missed for τ_3 When $U > 1.0$

Question 1.5

Assume that we instead of modifying $\tau_1$1, (set C1 = 2), want to increase the computation time for task τ_3 to be C3 = 17. Will all task complete before their deadlines? Draw a critical instant schedule for the given tasks. What is the system utilization bound?

The Utilization now is

$$U = \sum_{i=1}^{n} \frac{C_i}{T_i}$$
$$U = \frac{2}{10} + \frac{4}{15} + \frac{17}{35}$$
$$U = 0.9523810$$

As $U_{max,RM} \leq U \leq 1$, we cannot confirm if the tasks will miss their deadline or not simply by checking Utilization. For example, unlike in case 1.3 , even though we have similar conditions, we will have deadline miss in this scenario, as depicted in Figure 4.

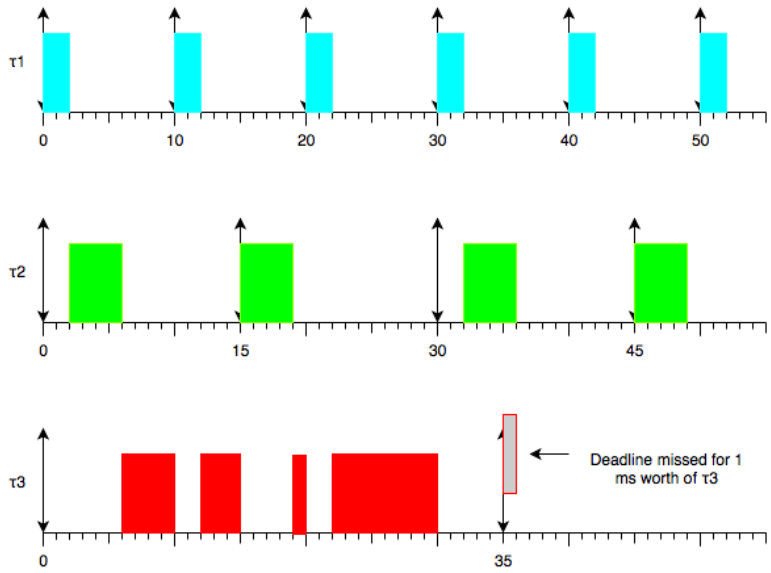


Figure 4: Schedule for (τ_1, τ_2, τ_3), Deadline Missed for τ_3, even though $U < 1.0$

Question 1.6

What conclusion can you draw from all this for the schedulability formula in Equation 2 on page 3? Specify your conclusion in terms of the system utilization bound, the $n(2^{1/n} - 1)$ ***expression and*** 1.00***.***

According to Liu and Layland (**Scheduling Algorithms for Multiprogramming in a Hard- Real-Time Environment, 1973**), the authors provided a analysis for schedulability of a Task set with implicit deadlines, which stated that for a Task Set T, with n tasks which cannot be blocked and suspend themselves, the Utilization Bound is a sufficient condition to ensure safe schedulability of the task set. However, if the bound is crossed, that is, the system utilization by the tasks is greater than its Utilization Bound, nothing can be said about the schedulability of the task set T.

Thus, we see in situation 1.2 (Figure 1), that when Utilization was less than Utilization Bound, the system was perfectly schedulable. However, in situation 1.3 (Figure 2), even though Utilization was greater than Utilization Bound, we still had a schedulable Task set. Then again, in situation 1.5 (Figure 4), the deadlines are missed for task τ_3 when Utilization is greater than Utilization Bound.

However, there still does exist a hard upper bound for a Task set with implicit deadlines on a single processor, which is simply complete 100% utilization of the processor. If a Task set exceeds this utilization, it will definitely miss deadlines for some of its tasks, and hence will not be schedulable. This is visible in situation 1.4 (Figure 3), where τ_3 missed its deadline.

Overall, we can summarize schedulability test by Utilization alone by the below formula (on a single processor with implicit deadline Tasks for RM Scheduling)

$$U = \sum_{i=1}^{n} \frac{C_i}{T_i} \quad \begin{cases} \leq n(2^{1/n} - 1) & \text{Task set is schedulable.} \\ > n(2^{1/n} - 1) \;\&\&\; \leq 1 & \text{May or may not be schedulable.} \\ > 1 & \text{Not schedulable.} \end{cases}$$

Thus, for the region $U_{max,RM} < U \leq 1$, we need another analysis technique to confirm if our Task set is schedulable or not.

Question 1.7

Insert the task set given in Figure 3 on page 3 in FpsCalc and calculate the response time of each task. What is the worst-case response time for each task using FpsCalc? Verify that the times correspond to the times you extracted using a critical instant schedule. In the worst case scenario, how many instances of τ_1 and τ_2 respectively can appear during one execution of τ_3 ? How does this values relate to the $\left\lceil \frac{R_i}{T_j} \right\rceil$ expression? How long time of the worst case response time of τ_3 is spent waiting for instances of τ_1 and τ_2 respectively?

From FpsCalc, we get the following values of response time for the Task set

τ_i	R_i
τ_1	2 ms
τ_2	6 ms
τ_3	24 ms

Below is the screen capture Figure 5 of FpsCalc Result (See Appendix 5 for code listing). :

```
sash2555@siegbahn:~/Desktop/FPSCALC$ ./fpscalc < exercise1_7.fps
This is fpscalc version 2.02 1997


System 'Assignment1_7'
----------------------

R[t1] = 2.000000
R[t2] = 6.000000
R[t3] = 24.000000
sash2555@siegbahn:~/Desktop/FPSCALC$
```

Figure 5: Worst Case Response Time, R_i for (τ_1, τ_2, τ_3), as calculated from FpsCalc

Again, observing Figure 1, we can see that the above calculated values match the worst case response time values for the Task set T on the first execution cycle for τ_1, τ_2, and τ_3 (The worst case execution time for a periodic implicit deadlines task set will be achieved for all its task when all of them are released at same time, and in this case, at $t = 0$).

Figure 6 shows worst case response time, in accordance with situation 1.2, as in Figure 1.

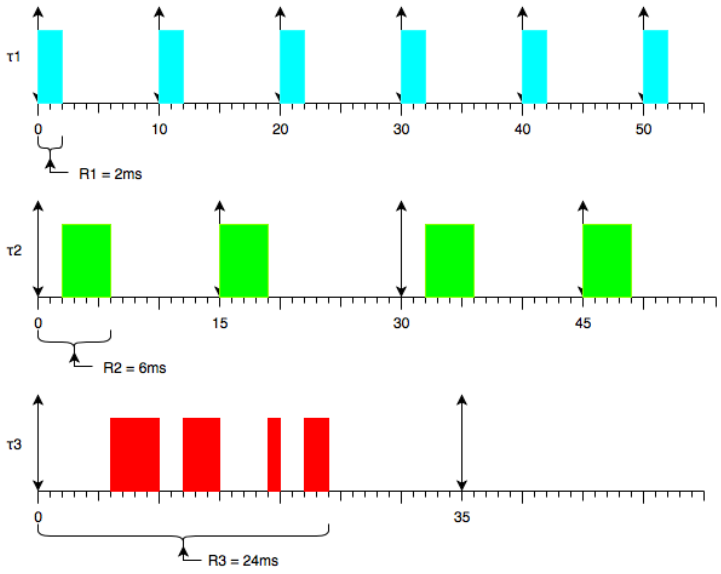


Figure 6: Worst Case Response Time, R_i for (τ_1, τ_2, τ_3), as observed from Figure 1

From the above Figure 6, we can see that during one execution instance of τ_3, we have two instances of τ_2 and three instances of τ_1.

The above values of number of occurances of τ_1 and τ_2 are equal to $\left\lceil \frac{R_i}{T_j} \right\rceil$ term in calculation of R_{τ_3}. The term signifies the number of times a higher priority task can preempt a lower priority task and finish with its worst case execution time. Thus, two occurances of τ_2 adds two times its C_2 to R_{τ_3}, while three occurances of τ_1 adds three times its own C_1 to R_{τ_3}. Thus, R_{τ_3} is made up of its own $C_3 = 10\ ms$, $3 \times C_1 = 3 \times 2 = 6\ ms$, and $2 \times C_2 = 2 \times 4 = 8\ ms$; Bringing $R_{\tau_3} = 10 + 6 + 8 = 24\ ms$.

2 Question 2

Task	C_i	T_i	D_i
τ_1	2 ms	20 ms	6 ms
τ_2	3 ms	7 ms	7 ms
τ_3	5 ms	14 ms	13 ms
τ_4	4 ms	100 ms	60 ms

Question 2.1

Given the task set in Figure 4 what is the priority ordering for the tasks using the DM priority ordering? What is the priority ordering using RM priority ordering? Use FpsCalc to calculate the response time for the tasks in both ordering. Will all tasks complete before their deadlines? If you are not convinced by the formulas you can do a critical instant schedule.

Deadline Monotonic Priority Ordering applies priorities to tasks in a task set according to their relative deadline; A task with shorter deadline gets higher priority. Thus, for the given task set, the DM priority order is:

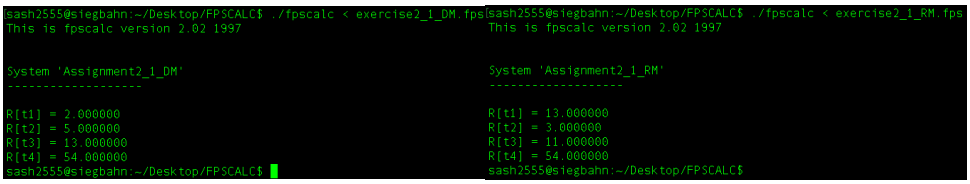
$$\tau_1 \prec \tau_2 \prec \tau_3 \prec \tau_4$$

On the other hand, if we hand out priorities according to Rate Monotonic Priority Ordering, we give higher priority to a task with shorter Period (or Rate). Thus, for the given task set, the RM priority order is:

$$\tau_2 \prec \tau_3 \prec \tau_1 \prec \tau_4$$

From FpsCalc, we get the following values of response time for the Task set for DM and RM priority ordering:

τ_i	D_i	$R_{i,\ RM}$	$R_{i,\ DM}$
τ_1	6 ms	13 ms	2 ms
τ_2	7 ms	3 ms	5 ms
τ_3	13 ms	11 ms	13 ms
τ_4	60 ms	54 ms	54 ms



(a) WCRT with DM Priority Scheduling (b) WCRT with RM Priority Scheduling

Figure 7: DM and RM Priority WCRTs

Figure 7 is the screen capture of FpsCalc Result(See Appendix 5 for code listing).

As we can see from the above results, with RM priority scheduling, τ_1 will have a scenario when its response time will be greater than its deadline, thus missing it. This scenario does not occur with DM priority scheduling. Thus, the task set will not be schedulable by RM Priority scheduling, however, it is possible to schedule it with DM Priority Scheduling.

Question 2.2

Sometimes it is preferable to not use strict RM or DM priority assignment when giving priorities to tasks. This can, for example, happen when we want to give a task with low deadline demands a better service rate or when the system is part of a larger distributed system.
Find two different priority assignments of the tasks in Figure 4 which are neither RM or DM and where deadlines are missed and met respectively.

We set Priority Ordering as

$$\tau_4 \prec \tau_3 \prec \tau_2 \prec \tau_1$$

Figure 8 shows the WCRTs as calculated from FpsCalc (See Appendix 5 for code listing).

```
sash2555@siegbahn:~/Desktop/FPSCALC$ ./fpscalc < exercise2_2_Unsuccessful.fps
This is fpscalc version 2.02 1997


System 'Assignment2_2_Unsuccessful'
-------------------

R[t1] = 28.000000
R[t2] = 12.000000
R[t3] = 9.000000
R[t4] = 4.000000
sash2555@siegbahn:~/Desktop/FPSCALC$
```

Figure 8: WCRTs as per FpsCalc (Deadlines Missed)

Now we set the Priority Ordering as

$$\tau_2 \prec \tau_1 \prec \tau_3 \prec \tau_4$$

This change brings the WCRTs as shown in Figure 9 as calculated from FpsCalc (See Appendix 5 for code listing).

```
sash2555@siegbahn:~/Desktop/FPSCALC$ ./fpscalc < exercise2_2_Successful.fps
This is fpscalc version 2.02 1997


System 'Assignment2_2_Successful'
--------------------

R[t1] = 5.000000
R[t2] = 3.000000
R[t3] = 13.000000
R[t4] = 54.000000
sash2555@siegbahn:~/Desktop/FPSCALC$
```

Figure 9: WCRTs as per FpsCalc (Deadlines Not Missed)

Summarizing, the above statements, the first priority ordering misses scheduled deadlines, while the second doesn't miss any deadlines, while neither are strictly RM or DM Priority Scheduling.

τ_i	D_i	$R_{i,\ Miss}$	$R_{i,\ No\ Miss}$
τ_1	6 ms	28 ms	5 ms
τ_2	7 ms	12 ms	3 ms
τ_3	13 ms	9 ms	13 ms
τ_4	60 ms	4 ms	54 ms

Question 2.3

Assume that we want to implement the tasks given in Figure 4 on a RT-kernel that only supports 3 priority levels and where tasks with the same priority will be handled in FIFO order by the scheduler. Assume that task τ_2 and τ_3 are set to have the same priority and that we use a DM priority assignment.
Define how the response time formula in Equation 3 on page 4 will be changed when we allow several tasks to have the same priority. Make sure that it is shown in your formula that a task, due to the FIFO order, might have to wait for one instance, but can't be preempted, of an equal priority task. How will the corresponding FpsCalc formula look like, (observe that sigma(ep, ...) includes the current task, τ_i)? What will now the worst case response time for each task be? Will all tasks meet their deadlines? Will the worst case response time for task τ_1 or τ_4 be affected? Conclusions?

The Worst Case Response time for a task τ_e with the given situations will occur when :

1. All tasks with priority equal as that of τ_e (including τ_e), are released at the same time.
2. Task τ_e gets to be at the end of the FIFO, that is, all the other tasks with equal priority as that of τ_e are already in the FIFO waiting to be executed.
3. All the other tasks exhibit their own worst-case response time.
4. As tasks with equal priorities cannot preempt each other, a task in the equal priority task set can be blocked only once per execution by another task in the same set, thus being held from execution for a time equal to the worst-case execution time of the blocking equal priority task.

From the above discussion, we can confirm that the response time R_i of a task τ_i will be the sum of the following terms :

- Preemption time by tasks with higher priority, with number of preemptions given by $\left\lceil \frac{R_i}{T_j} \right\rceil$, where T_j is the period of a higher priority task. Thus with worst case execution time of C_j of each higher priority task, τ_i will be blocked for $\left\lceil \frac{R_i}{T_j} \right\rceil * C_j$ time units for each such higher priority task.
- Blocked by equal priority tasks with worst case execution time of C_k. In the worst case, when all equal priority tasks are ahead of τ_i in the FIFO, the worst case blocking time would be $\sum_{k \in ep(\tau_i)} C_k$ time units.

Thus the worst case response time formula for task τ_i would be given by

$$R_i = \sum_{k \in ep(\tau_i)} C_k + \sum_{j \in hp(\tau_i)} \left\lceil \frac{R_i}{T_j} \right\rceil * C_j$$

Figure 10 shows the FpsCalc result(See Appendix 5 for Code Listing) with the above formual for the task set, with priority order as

$$\tau_1 \prec \tau_2 = \tau_3 \prec \tau_4$$

```
sash2555@siegbahn:~/Desktop/FPSCALC$ ./fpscalc < exercise2_3.fps
This is fpscalc version 2.02 1997


System 'Assignment2_3'
--------------------

R[t1] = 2.000000
R[t2] = 10.000000
R[t3] = 10.000000
R[t4] = 54.000000
sash2555@siegbahn:~/Desktop/FPSCALC$
```

Figure 10: WCRTs as per FpsCalc

τ_i	D_i	R_i
τ_1	6 ms	2 ms
τ_2	7 ms	10 ms
τ_3	13 ms	10 ms
τ_4	60 ms	54 ms

As τ_2 has its R_i greater than its deadline D_i, τ_2 will miss its deadline.

We can see that τ_1 and τ_4 have the same worst-case response times as before with simple DM priority ordering with four different priorities. This is because, for τ_1, all other tasks are still of lower priority than itself, hence it does not suffer any change in its response time because of change in their priority. Similarly, for τ_4, all other tasks are still at a higher priority than itself and can and will preempt it as before, resulting in the same worst-case response time.

3 Question 3

Task	C_i	T_i	D_i
τ_1	2 ms	10 ms	5 ms
τ_2	3 ms	20 ms	12 ms
τ_3	10 ms	40 ms	40 ms
τ_4	4 ms	100 ms	50 ms

Question 3.1

Assume that we have the tasks shown in Figure 6. Give priorities to the tasks according to the DM priority assignment. Will all tasks always meet their deadlines?

As per DM Priority scheduling, priority order would be

$$Priority(\tau_1) > Priority(\tau_2) > Priority(\tau_3) > Priority(\tau_4)$$

As per FpsCalc, the WCRTs are as shown in screen capture in Figure 11 (See Appendix 5 for Code Listing) .

```
[sash2555@siegbahn:~/Desktop/FPSCALC$ ./fpscalc < exercise3_1.fps
This is fpscalc version 2.02 1997


System 'Assignment3_1'
--------------------

R[t1] = 2.000000
R[t2] = 5.000000
R[t3] = 17.000000
R[t4] = 26.000000
sash2555@siegbahn:~/Desktop/FPSCALC$
```

Figure 11: WCRTs as per FpsCalc

τ_i	D_i	R_i
τ_1	5 ms	2 ms
τ_2	12 ms	5 ms
τ_3	40 ms	17 ms
τ_4	50 ms	26 ms

As all response times are lesser than the deadlines for the respective tasks, the task set would be schedulable.

Question 3.2

Assume that task τ_2 and τ_4 in Figure are sharing a semaphore $S1$ and that τ_2 and τ_4 executes for at most $1\ ms$ and $2\ ms$ respectively in the critical section. Show, by doing a critical instant scheme that the deadline for task τ_2 can be missed if semaphores and no mechanism for limiting priority inversion (see below) is used. Tip: You might have to model that τ_4 has taken the semaphore, just before the moment where you let all the other tasks start executing at the same time.

From Figure 12, we can see that with no way to prevent priority inversion, τ_2 can miss its deadline.

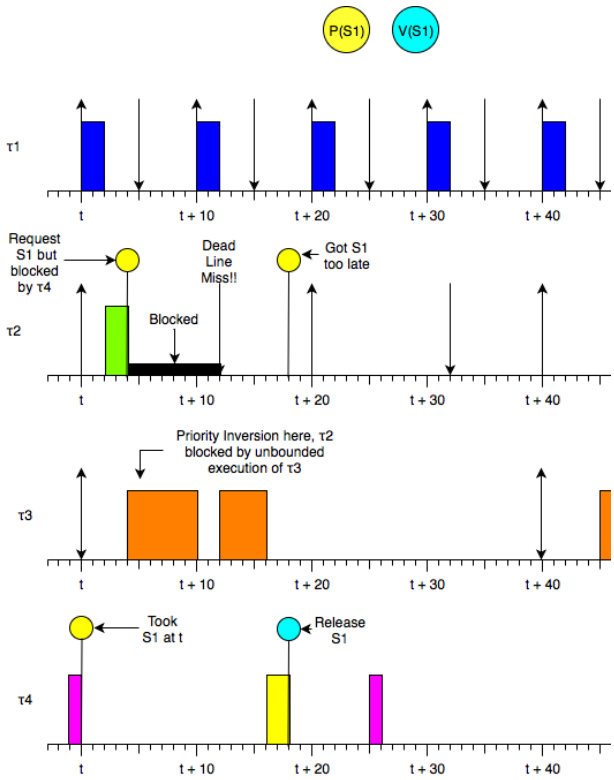


Figure 12: τ_2 Missing Deadline Due to Unbounded Priority Inversion from task τ_3

Question 3.3

Assume that task τ_2 not only is sharing a semaphore $S1$ with τ_4 but also is sharing a semaphore $S2$ with task τ_3 as given in Figure 7. Also assume that the times the task τ_2 is accessing the semaphores $S1$ and $S2$ do not overlap. Show, by doing a critical instant scheme that deadline for task τ_2 can be missed even though the priority inheritance protocol is used.

From Figure 13, we can see that even with Basic Priority Inversion Protocol,, τ_2 can miss its deadline.

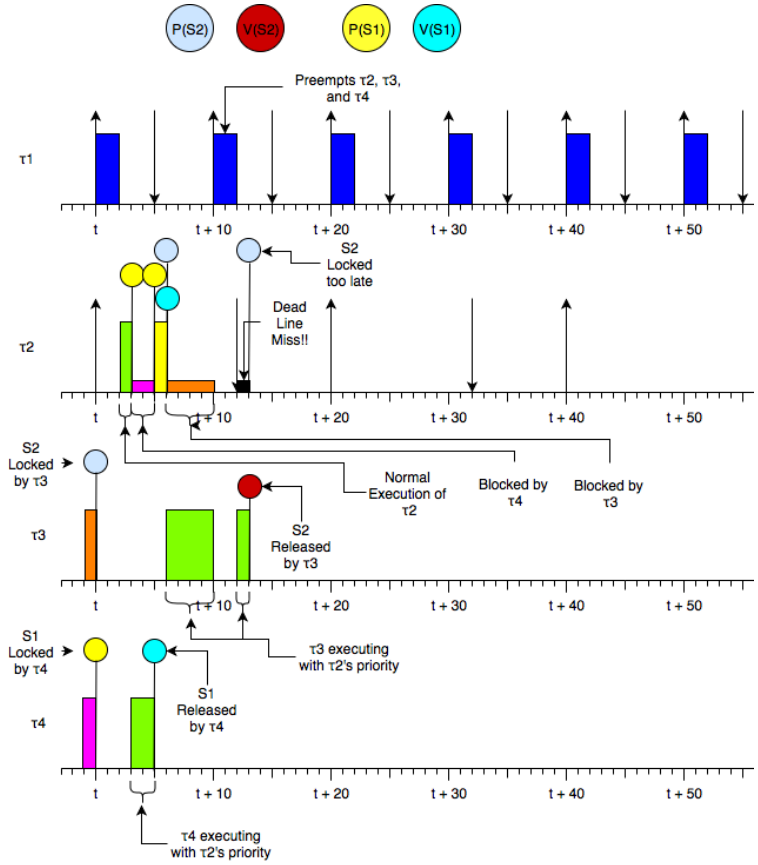


Figure 13: τ_2 Missing Deadline Due to Chained blocking of τ_2 from τ_3 and τ_4

Question 3.4

What are the blocking times for the tasks in Figure 6 using the priority inheritance protocol? Make sure that the amount of blocking corresponds to your drawn schedule. Tip: several tasks will experience blocking. What are the response times for the tasks when using the priority inheritance protocol?

τ_i	B_i
τ_1	0 ms
τ_2	7 ms
τ_3	2 ms
τ_4	0 ms

As τ_1 has no shared resources with any of the other tasks, its blocking time is 0 ms. Task τ_2 uses both shared resources, and hence can get blocked by the sum of maximum times a lower priority (that τ_2) locks the semaphore, which is, (sum of $cs_{\tau_3,s_2} + cs_{\tau_4,s_1} = 5 + 2 =$) 7 ms. Similarly, τ_3 can be blocked as it can be made to do so when τ_4 is executing with lock on S_1 and executing with priority of τ_2. Thus, the blocking time for τ_3 would be ($cs_{\tau_4,s_1} =$) 2 ms. Task τ_4 which is the lowest priority task, will not suffer from any blocking, and hence will have a blocking time of 0 ms.

Question 3.5

What are the priority ceilings for the semaphores S_1 and S_2? What are the blocking times for the tasks in Figure 6 using the immediate inheritance protocol? Tip: several tasks will experience blocking. Explain why task τ_3 can experience blocking even though it does not share any semaphore with τ_4. Will all tasks complete before their deadlines?

Semaphore S_i	**Task set shared by S_i**	**Highest Priority in Task Set**	$Ceil(S)$
S_1	$\{\tau_2,\ \tau_4\}$	$max(pri(\tau_2),\ pri(\tau_4)) = pri(\tau_2) = 2$	2
S_2	$\{\tau_2,\ \tau_3\}$	$max(pri(\tau_2),\ pri(\tau_3)) = pri(\tau_2) = 2$	2

For the above table, we can see that both semaphores have a priority ceiling of 2.

For calculating the Blocking times for the task set, we use the formula

$$B_i = \max_{\{\ k,s \mid k \in lp(i) \ \wedge\ s \in used\_by(k) \ \wedge\ ceil(s) \geq pri(i)\ \}} [cs_{k,s}]$$

Using the above formula, we calculate B_i as per the below table.

τ_i	$P(\tau_i)$	$\{k\}$	$\{s\}$	$\{ceil(s) \geq pri(i)\}$	$\{cs_{k,s}\}$	B_i
τ_1	1	$\{\tau_2, \tau_3, \tau_4\}$	$\{S_{1,\ (CP=2)},\ S_{2,\ (CP=2)}\}$	$\{\phi\}$	$\{\phi\}$	0 ms
τ_2	2	$\{\tau_3, \tau_4\}$	$\{S_{1,\ (CP=2)},\ S_{2,\ (CP=2)}\}$	$\{S_1,\ S_2\}$	$\{2,\ 5\}$	5 ms
τ_3	3	$\{\tau_4\}$	$\{S_{1,\ (CP=2)}\}$	$\{S_1\}$	$\{2\}$	2 ms
τ_4	4	$\{\phi\}$	$\{\phi\}$	$\{\phi\}$	$\{\phi\}$	0 ms

We see from above results, that τ_3 suffers from a 2 ms blocking time. This is because, it shares a resource(S_2) with task τ_2, which in turn shares another resource(S_1) with task τ_4. Thus it can happen, that τ_4 can get a lock on S_1, and inherits the priority of τ_2, preventing τ_3 from execution. As the length of critical section for execution of S_1 by τ_4 is 2 ms, thus, τ_3 suffers from a 2 ms blocking time.

From FpsCalc, we have the following result for WCRTs as shown in Figure 14 (See Appendix 5 for Code Listing).

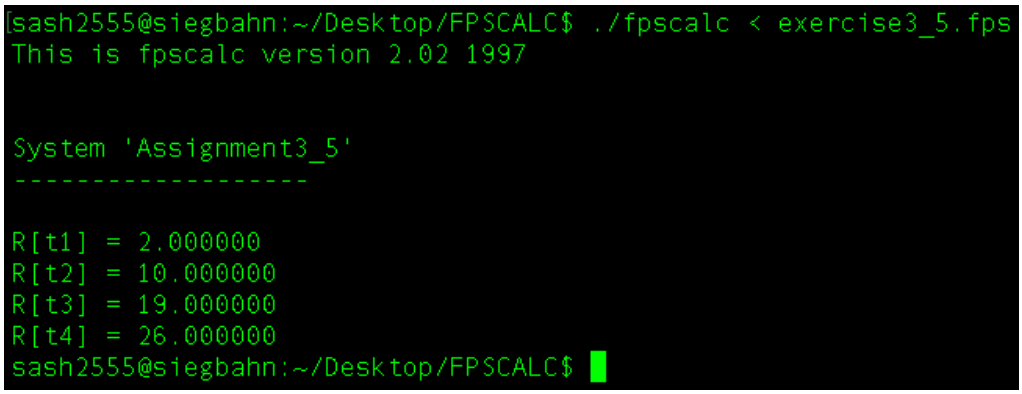


Figure 14: WCRTs as per FpsCalc

τ_i	D_i	R_i
τ_1	5 ms	2 ms
τ_2	12 ms	10 ms
τ_3	40 ms	19 ms
τ_4	50 ms	26 ms

As all response times are lesser than the deadlines for the respective tasks, the task set would be schedulable.

4 Question 4

Question 4.1

Another source of jitter is varying execution and response times for tasks (or messages) that start other tasks. To illustrate this assume a system with two tasks τ_1 ***and*** τ_2***. Let task*** τ_1 ***have a period*** $T_1 = 10$ ***and a fixed (non-varying) execution time*** $C_1 = 3$***. Let task*** τ_2 ***arrives always*** 2 ***time units after*** τ_1 ***arrives, but it waits for*** τ_1 ***finishing before it gets released (we can see it as that*** τ_2 ***waits for the result of*** τ_1 ***before it can execute). Assume task*** τ_1 ***always ends its execution by releasing task*** τ_2 ***(*** τ_2 ***can now be scheduled for execution). Let*** τ_2 ***have a worst case execution time of*** $C_2 = 2$***.***
Draw a schedule that shows two instances of τ_1 ***and*** τ_2 ***respectively. What is the period*** T_2 ***of task*** τ_2***?***

Figure 15 shows instances of τ_1 and τ_2.

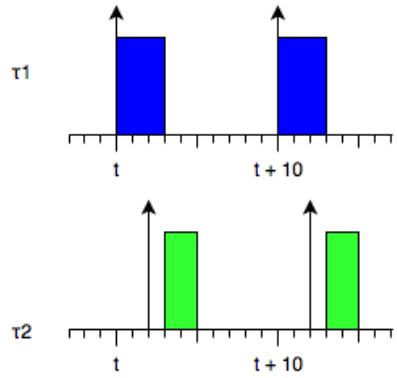


Figure 15: Instance of τ_1 and τ_2

As τ_2 arrives after a constant time period from the arrival of τ_1, hence the period of task τ_2, T_2, is same as that of task τ_1, that is, 10 ms.

Question 4.2

To illustrate that varying execution time of τ_1 might cause jitter of τ_2 assume that τ_1's execution time no longer is fixed but varies between ${C_1}^{min} = 3$ and ${C_1}^{max} = 5$. Task τ_1 still starts task τ_2 at the end of its execution.
Draw a schedule with two instances of τ_1 that shows that the varying execution time of τ_1 might give raise to jitter of τ_2. What is the jitter that task τ_2 can experience?

Figure 16 shows instances of τ_1 and τ_2.

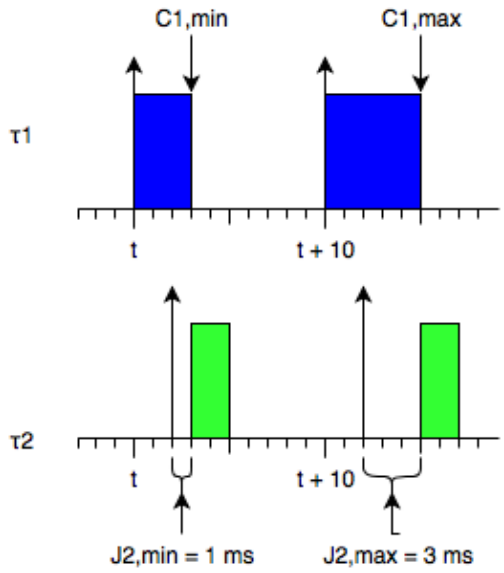


Figure 16: Instance of τ_1 and τ_2

The release jitter that can be suffered by τ_2 is the difference between the earliest and the latest time τ_2 can be released, with respect to its invocation time. Thus, from the Figure 16, we can see that the earliest τ_2 is released is at 13 ms, while the latest release is at 15 ms. Therefore, the jitter is $(15 - 13 =)$ 2 ms.

Question 4.3

To illustrate that interference of high priority tasks might give raise to further jitter of low priority tasks we add a task τ_0 ***to the system. Let task*** τ_0 ***have higher priority than both*** τ_1 ***and*** τ_2***, a period*** $T_0 = 20$ ***and a worst case execution time*** $C_0 = 2$***.***
Draw a schedule that shows that varying response time of τ_1 ***due to interference of*** τ_0 ***will give raise to further jitter of*** τ_2 ***. Task*** τ_1***'s execution time still varies between*** ${C_1}^{min}$ ***and*** ${C_1}^{max}$***. What is the jitter that*** τ_2 ***can experience due to varying response and execution time of*** τ_1***?***

Figure 17 shows instances of τ_0, τ_1, and τ_2.

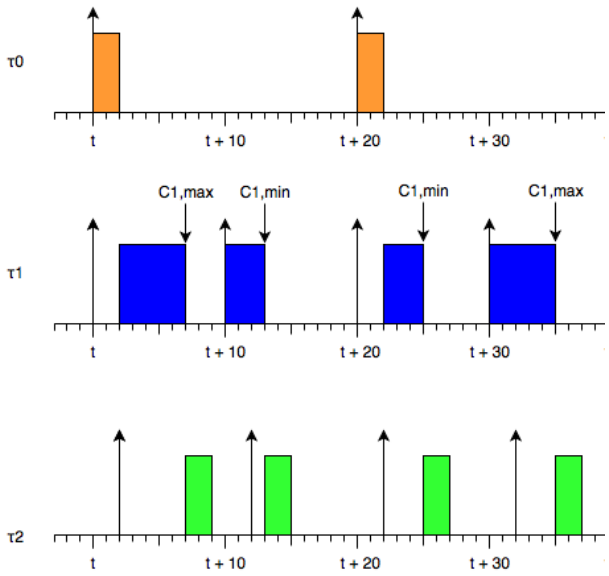


Figure 17: Instance of τ_0, τ_1, and τ_2

Due to introduction of higher priority task τ_0, τ_2 will now suffer greater maximum release Jitter, ${J_2}^{Biggest} = 5\ ms$(During the very first instance of schedule in Figure 17). This will change the release jitter again, which now becomes $(5 - 1 =)\ 4\ ms$ (as ${J_2}^{Smallest} = 1\ ms$ remains the same).

Question 4.4

Task	C_i	T_i	D_i	J_i
τ_A	5 ms	20 ms	10 ms	5 ms
τ_B	30 ms	50 ms	50 ms	10 ms

For the given tasks τ_A and τ_B with DM priority ordering, what is the worst case response time for respective task assuming that we have no jitter. Will both tasks be able to complete before their deadlines?

As per FpsCalc, when ignoring any jitter, we get the following WCRTs for τ_A and τ_B as shown in Figure 18 (See Appendix 5 for Code Listing).

```
sash2555@siegbahn:~/Desktop/FPSCALC$ ./fpscalc < exercise4_4.fps
This is fpscalc version 2.02 1997


System 'Assignment4_4'
--------------------

R[tA] = 5.000000
R[tB] = 40.000000
sash2555@siegbahn:~/Desktop/FPSCALC$
```

Figure 18: WCRTs as per FpsCalc

τ_i	D_i	R_i
τ_A	10 ms	5 ms
τ_B	50 ms	40 ms

As we can see from above table, as both tasks have their R_i less than their respective D_i, the tasks will complete before their deadlines.

Question 4.5

Given the formula in Equation 5 what is the worst case response time for respective tasks assuming that they can experience jitter? Will both tasks be able to complete before their deadlines?

As per FpsCalc, without ignoring jitter, we get the following W_i and R_i for τ_A and τ_B as shown in Figure 19 (See Appendix 5 for Code Listing).

```
sash2555@siegbahn:~/Desktop/FPSCALC$ ./fpscalc < exercise4_5.fps
This is fpscalc version 2.02 1997


System 'Assignment4_4'
----------------------

W[tA] = 5.000000
W[tB] = 45.000000

R[tA] = 10.000000
R[tB] = 55.000000
sash2555@siegbahn:~/Desktop/FPSCALC$
```

Figure 19: WCRTs as per FpsCalc

τ_i	D_i	R_i
τ_A	10 ms	10 ms
τ_B	50 ms	55 ms

As from the above data, we can see τ_A has its worst case response time lesser than its deadline, therefore it will not miss any deadlines. However, τ_B has a worst case response time of 55 ms, which is greater than its deadline of 50 ms. This will result in τ_B missing its deadline.

Question 4.6

Draw a task schedule for the tasks τ_A and τ_B which gives the same worst case response times as the formula in Equation 5. Indicate arrival, jitter, beginning of execution, execution, preemption, and completion for each task in your schedule.

Figure 20 shows the task schedule for τ_A and τ_B, with the later missing its deadline.

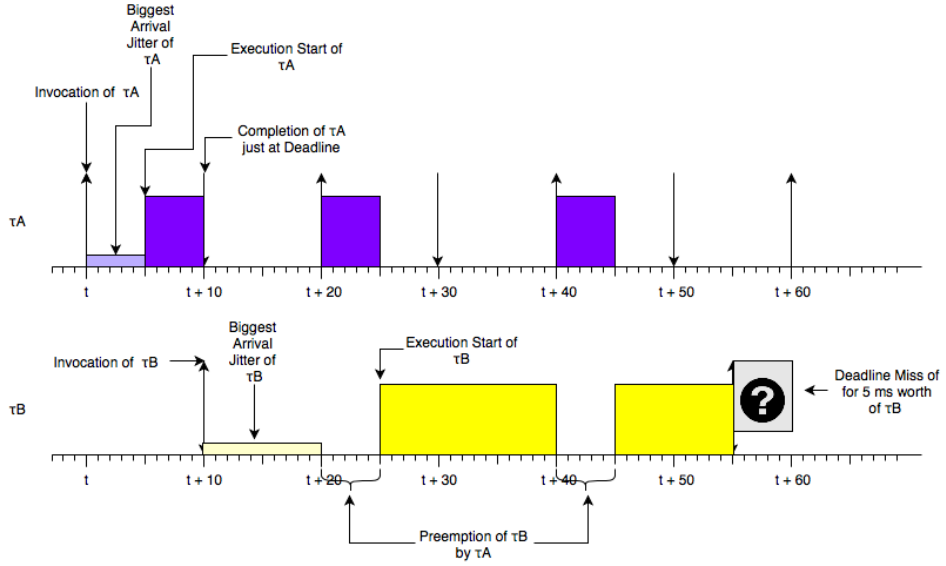


Figure 20: Schedule for τ_A and τ_B with Jitter

5 Appendix

Code Listing for Question 1.7

```
system Assignment1_7 {

  declarations {
    tasks t1, t2, t3;
    indexed T,C,R,D,U;
    priority P;
  }

  initialise {

    ! Periods
    T[t1] = 10;
    T[t2] = 15;
    T[t3] = 35;

    ! WCETs
    C[t1] = 2;
    C[t2] = 4;
    C[t3] = 10;

    ! Deadlines (not used)
    D[t1] = 10;
    D[t2] = 15;
    D[t3] = 35;

    ! Priorities
    P[t1] = 1;
    P[t2] = 2;
    P[t3] = 3;
  }

   formulas {

    ! Calculate the response-time for each task
    R[i] = C[i] + sigma(hp, ceiling((R[i])/T[j]) * C[j]);
  }
}
```

Code Listing for Question 2.1, DM and RM Scheduling

DM Listing

```
system Assignment2_1_DM {

  declarations {
    tasks t1, t2, t3, t4;
    indexed T,C,R,D,U;
    priority P;
  }

  initialise {

    ! Periods
    T[t1] = 20;
    T[t2] = 7;
    T[t3] = 14;
    T[t4] = 100 ;

    ! WCETs
    C[t1] = 2;
    C[t2] = 3;
    C[t3] = 5;
    C[t4] = 4;

    ! Deadlines
    D[t1] = 6;
    D[t2] = 7;
    D[t3] = 13;
    D[t4] = 60;

    ! Priorities as per DM
    P[t1] = 1;
    P[t2] = 2;
    P[t3] = 3;
    P[t4] = 4;
  }

   formulas {

    ! Calculate the response-time for each task
    R[i] = C[i] + sigma(hp, ceiling((R[i])/T[j]) * C[j]);
  }
}
```

RM Listing

```
system Assignment2_1_RM {

  declarations {
    tasks t1, t2, t3, t4;
    indexed T,C,R,D,U;
    priority P;
  }

  initialise {

    ! Periods
    T[t1] = 20;
    T[t2] = 7;
    T[t3] = 14;
    T[t4] = 100 ;

    ! WCETs
    C[t1] = 2;
    C[t2] = 3;
    C[t3] = 5;
    C[t4] = 4;

    ! Deadlines
```

```
24      D[t1] = 6;
25      D[t2] = 7;
26      D[t3] = 13;
27      D[t4] = 60;
28
29      ! Priorities as per RM
30      P[t1] = 3;
31      P[t2] = 1;
32      P[t3] = 2;
33      P[t4] = 4;
34    }
35
36     formulas {
37
38      ! Calculate the response-time for each task
39      R[i] = C[i] + sigma(hp, ceiling((R[i])/T[j]) * C[j]);
40    }
41  }
```

Code Listing for Question 2.2, Unsuccessful and Successful Scheduling

Unsuccessful Scheduling

```
system Assignment2_2_Unsuccessful {

  declarations {
    tasks t1, t2, t3, t4;
    indexed T,C,R,D,U;
    priority P;
  }

  initialise {

    ! Periods
    T[t1] = 20;
    T[t2] = 7;
    T[t3] = 14;
    T[t4] = 100 ;

    ! WCETs
    C[t1] = 2;
    C[t2] = 3;
    C[t3] = 5;
    C[t4] = 4;

    ! Deadlines
    D[t1] = 6;
    D[t2] = 7;
    D[t3] = 13;
    D[t4] = 60;

    ! Priorities
    P[t1] = 4;
    P[t2] = 3;
    P[t3] = 2;
    P[t4] = 1;
  }

   formulas {

    ! Calculate the response-time for each task
    R[i] = C[i] + sigma(hp, ceiling((R[i])/T[j]) * C[j]);
  }
}
```

Successful Scheduling

```
system Assignment2_2_Successful {

  declarations {
    tasks t1, t2, t3, t4;
    indexed T,C,R,D,U;
    priority P;
  }

  initialise {

    ! Periods
    T[t1] = 20;
    T[t2] = 7;
    T[t3] = 14;
    T[t4] = 100 ;

    ! WCETs
    C[t1] = 2;
    C[t2] = 3;
    C[t3] = 5;
    C[t4] = 4;

    ! Deadlines
```

```
24      D[t1] = 6;
25      D[t2] = 7;
26      D[t3] = 13;
27      D[t4] = 60;
28
29      ! Priorities
30      P[t1] = 2;
31      P[t2] = 1;
32      P[t3] = 3;
33      P[t4] = 4;
34    }
35
36     formulas {
37
38      ! Calculate the response-time for each task
39      R[i] = C[i] + sigma(hp, ceiling((R[i])/T[j]) * C[j]);
40    }
41  }
```

Code Listing for Question 2.3

```
system Assignment2_3 {

  declarations {
    tasks t1, t2, t3, t4;
    indexed T,C,R,D,U;
    priority P;
  }

  initialise {

    ! Periods
    T[t1] = 20;
    T[t2] = 7;
    T[t3] = 14;
    T[t4] = 100 ;

    ! WCETs
    C[t1] = 2;
    C[t2] = 3;
    C[t3] = 5;
    C[t4] = 4;

    ! Deadlines
    D[t1] = 6;
    D[t2] = 7;
    D[t3] = 13;
    D[t4] = 60;

    ! Priorities
    P[t1] = 1;
    P[t2] = 2;
    P[t3] = 2;
    P[t4] = 3;
}

   formulas {

    ! Calculate the response-time for each task
    R[i] =sigma(ep,C[j]) + sigma(hp, ceiling(R[i]/T[j]) * C[j]);

  }
}
```

Code Listing for Question 3.1

```
system Assignment3_1 {

  declarations {
    tasks t1, t2, t3, t4;
    indexed T,C,R,D,U;
    priority P;
  }

  initialise {

    ! Periods
    T[t1] = 10;
    T[t2] = 20;
    T[t3] = 40;
    T[t4] = 100 ;

    ! WCETs
    C[t1] = 2;
    C[t2] = 3;
    C[t3] = 10;
    C[t4] = 4;

    ! Deadlines
    D[t1] = 5;
    D[t2] = 12;
    D[t3] = 40;
    D[t4] = 50;

    ! Priorities
    P[t1] = 1;
    P[t2] = 2;
    P[t3] = 3;
    P[t4] = 4;
}

   formulas {

    ! Calculate the response-time for each task
    R[i] = C[i] + sigma(hp, ceiling((R[i])/T[j]) * C[j]) ;
  }
}
```

Code Listing for Question 3.5

```
system Assignment3_5 {

  declarations {
    tasks t1, t2, t3, t4;
    indexed T,C,R,D,U,B;
    priority P;
  }

  initialise {

    ! Periods
    T[t1] = 10;
    T[t2] = 20;
    T[t3] = 40;
    T[t4] = 100 ;

    ! WCETs
    C[t1] = 2;
    C[t2] = 3;
    C[t3] = 10;
    C[t4] = 4;

    ! Deadlines
    D[t1] = 5;
    D[t2] = 12;
    D[t3] = 40;
    D[t4] = 50;

    ! Priorities
    P[t1] = 1;
    P[t2] = 2;
    P[t3] = 3;
    P[t4] = 4;

    ! Blocking times as calculated as per priority ceiling protocol
    B[t1] = 0 ;
    B[t2] = 5 ;
    B[t3] = 2 ;
    B[t4] = 0 ;
}

   formulas {

    ! Calculate the response-time for each task
    R[i] = C[i] + B[i] + sigma(hp, ceiling((R[i])/T[j]) * C[j]) ;
 }
}
```

Code Listing for Question 4.4

```
system Assignment4_4 {

  declarations {
    tasks tA, tB ;
    indexed T,C,R,D;
    priority P;
  }

  initialise {

    ! Periods
    T[tA] = 20;
    T[tB] = 50;

    ! WCETs
    C[tA] = 5;
    C[tB] = 30;

    ! Deadlines
    D[tA] = 10;
    D[tB] = 50;

    ! Priorities
    P[tA] = 1;
    P[tB] = 2;

}

   formulas {

    ! Calculate the response-time for each task
    R[i] = C[i]  + sigma(hp, ceiling((R[i])/T[j]) * C[j]) ;

  }
}
```

Code Listing for Question 4.5

```
system Assignment4_4 {

  declarations {
    tasks tA, tB ;
    indexed T,C,R,D,J,W;
    priority P;
  }

  initialise {

    ! Periods
    T[tA] = 20;
    T[tB] = 50;

    ! WCETs
    C[tA] = 5;
    C[tB] = 30;

    ! Deadlines
    D[tA] = 10;
    D[tB] = 50;

    ! Priorities
    P[tA] = 1;
    P[tB] = 2;

   ! Jitter
   J[tA] = 5 ;
   J[tB] = 10 ;

}

   formulas {

    ! Calculate the response-time for each task
    W[i] = C[i]  + sigma(hp, (1 + ceiling((W[i] - (T[j] - J[j]) )/T[j])) * C[
        j]) ;
    R[i] = W[i] + J[i] ;
  }
}
```