Name: SOLUTIONS **(Bonus: 1 point)**

Andrew ID: *solve648* **(Bonus: 1 point)**

18-648: Embedded Real-Time Systems

Quiz #1 Solutions

Fall 2016

60 minutes

Instructions

1. Show all relevant work.
2. Partial credit may be given for some questions.
3. The use of a calculator is allowed.
4. The time limit will be *strictly* enforced.
5. Be efficient. Come back to questions you are not sure of later.
6. <u>Watch the screen for any clarifications</u>.
7. This is a **CLOSED-BOOK/CLOSED-NOTES quiz**.

For Graders' Use Only

Bonus: _________ / 2

1._________ / 22

2._________ / 24

3._________ / 22

4._________ / 20

5._________ / 12

TOTAL. _________ / 100

Question 1. True/False and Multiple Choices.

(a) True/False Questions (18 points)

False Using the Priority Ceiling Protocol (PCP) can lead to mutual deadlocks.

True It takes a taskset with one shared resource and at least three or more tasks to exhibit unbounded priority inversion.

False Consider a fixed-priority scheduler which assigns random (fixed) priority values to periodic real-time tasks. There might be priority assignments on some task set for which this scheduler would find a feasible schedule but the rate-monotonic scheduler will not.

False Earliest deadline first (EDF) is a static priority preemptive scheduling scheme.

True There are periodic task sets that can be scheduled by EDF but not RMS.

False There are periodic task sets that can be scheduled by RMS but not EDF.

False With a priority-based preemptive scheduler, a ready task scheduled to run will run until its completion, after which the highest-priority ready task on the ready queue will be run.

False Each thread in a process has its own data segment.

True A taskset with three periodic tasks that have periods 8, 16 and 48 time-units respectively is considered harmonic.

True Context-switching which consumes a non-zero amount of time can be accounted for in schedulability analysis.

(b) Circle one or more answers for the questions below. (4 points)

I. Which best describes the EDF scheduler used by real-time tasks:

a. Static preemptive

b. Static non-preemptive

c. **Dynamic preemptive**

d. Dynamic non-preemptive

II. Which of the following protocols do not provide deadlock avoidance (assuming tasks do not suspend within critical regions):

a. **Basic Priority Inheritance Protocol**

b. Highest Locker Priority Protocol

c. Priority Ceiling Protocol

d. Non-Preemption Protocol

Question 2. Will it schedule? (24 points)

For the following task sets, determine if the set listed is schedulable using Rate-Monotonic Scheduling. Each task τ_i is characterized by $\{C_i, T_i\}$, its worst-case execution time and period.. The relative deadline of each task is the same as its period. If a task set is not schedulable, please indicate which task misses its deadline. The RMS bound is $n(2^{1/n} - 1)$. For n = 2, U_b = 0.828. For n = 3, U_b =0.78. Use the response-time test if appropriate. The response-time test for task i is given by:

$$a^i_{k+1} = C_i + \sum_{j=1}^{i-1} \left\lceil \frac{a^i_k}{T_j} \right\rceil C_j \qquad \text{where } a^i_0 = \sum_{j=1}^{i} C_j$$

Test terminates when $a^i_{k+1} = a^i_k$

a) *Hint*: Remember to use RMS.

$\tau_1 = \{64, 160\}$
$\tau_2 = \{10, 80\}$
$\tau_3 = \{8, 32\}$

Since RMS being used, let's re-arrange the tasks in RMS priority order.

$\tau'_1 = \{8, 32\}$ → $U'_1 = \frac{8}{32} = 0.25$
$\tau'_2 = \{10, 80\}$ → U'_2 = 10/80 = 0.125
$\tau'_3 = \{64, 160\}$ → U'_3 = 64 / 160 = 0.40

U_{total} = 0.25 + 0.125 + 0.40 = 0.775

Since $U_{bound}(3)$ = 0.78, we can immediately declare that **all 3 tasks are schedulable** under RMS.

b) *Hint*: Remember to use RMS.

$\tau_1 = \{3, 12\}$
$\tau_2 = \{6, 20\}$
$\tau_3 = \{6, 15\}$

Since RMS being used, let's re-arrange the tasks in RMS priority order.

$\tau'_1 = \{3, 12\}$ $\rightarrow U'_1 = \frac{3}{12} = 0.25$
$\tau'_2 = \{6, 15\}$ $\rightarrow U'_2 = 6/15 = 0.4$
$\tau'_3 = \{6, 20\}$ $\rightarrow U'_3 = 6 / 20 = 0.30$

$U_{total} = 0.25 + 0.4 + 0.30 = 0.95$

Since $U_{bound}(3) = 0.78 < U_{total}$, the schedulability bound cannot be used to determine the schedulability of the entire taskset. We therefore have to analyze the tasks in descending priority order.

Task τ'_1: the task is schedulable since $U'_1 = 0.25 \leq U_{bound}(1) = 1.0$.

Task τ'_2: we need to check if $U'_1 + U'_2 = 0.25 + 0.4 \leq U_{bound}(2) = 0.828$.
Yes, the task is schedulable.

Task τ'_3: when we check if $U'_1 + U'_2 + U'_3 = 0.25 + 0.4 + 0.3 \leq U_{bound}(3) = 0.78$, we find that the test is not passed. So, we perform the exact completion time for τ'_3.

$$a^i_{k+1} = C_i + \sum_{j=1}^{i-1} \left\lceil \frac{a^i_k}{T_j} \right\rceil C_j \qquad \text{where } a^i_0 = \sum_{j=1}^{i} C_j \qquad \boxed{i = 3}$$

Test terminates when $a^i_{k+1} = a^i_k$

$a^3_0 = C'_1 + C'_2 + C'_3 = 3 + 6 + 6 = 15$
$a^3_1 = C'_3 + \text{ceiling}(15/12)*3 + \text{ceiling}(15/15)*6 = 6 + 2*3 + 1*6 = 18$
$a^3_2 = C'_3 + \text{ceiling}(18/12)*3 + \text{ceiling}(18/15)*6 = 6 + 2*3 + 2*6 = 24 > T'_3$

Since $a^3_2 = 24$, which is already greater than 20 (the deadline of τ'_3), we can stop. Task τ'_3 is **not** schedulable.

Question 3. Sharing is caring (22 points)

Suppose each periodic real-time task is represented by (C, T=D).
Consider the 4-task set τ_1: {4, 12}, τ_2: {3, 16}, τ_3: {3, 20}, τ_4: {3, 24}.
The tasks arrive in following manner:

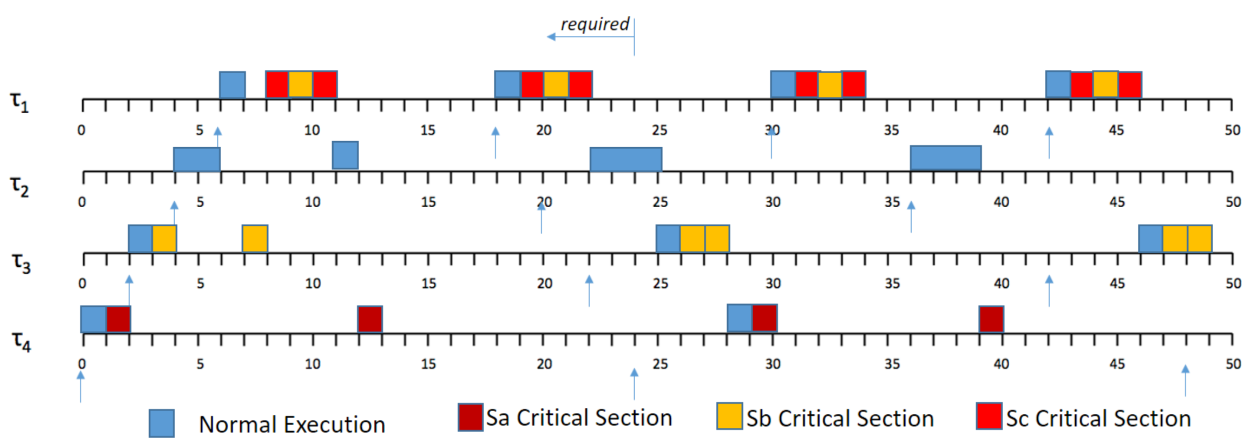
1. τ_4 at time 0
2. τ_3 at time 2
3. τ_2 at time 4
4. τ_1 at time 6

And they execute in the following manner:

1. τ_4 executes normally for 1 time unit, locks mutex S_a, executes for 2 time units and finally unlocks mutex S_a. (....P(S_a)....V(S_a))
2. τ_3 executes normally for 1 sec, locks mutex S_b, executes for 2 time units and finally unlocks mutex S_b. (....P(S_b)....V(S_b))
3. τ_2 execute normally for 3 time units. (....)
4. τ_1 executes normally for 1 time unit, locks mutex S_c, executes for 1 more time unit, locks mutex S_b executes for 1 time unit, unlocks mutex S_b, executes for 1 time unit and finally unlocks mutex S_c. (....P(S_c)....P(S_b)....V(S_b)....V(S_c))

a) Draw the timeline under RMS using Priority Ceiling Protocol. Please mark all the critical sections.

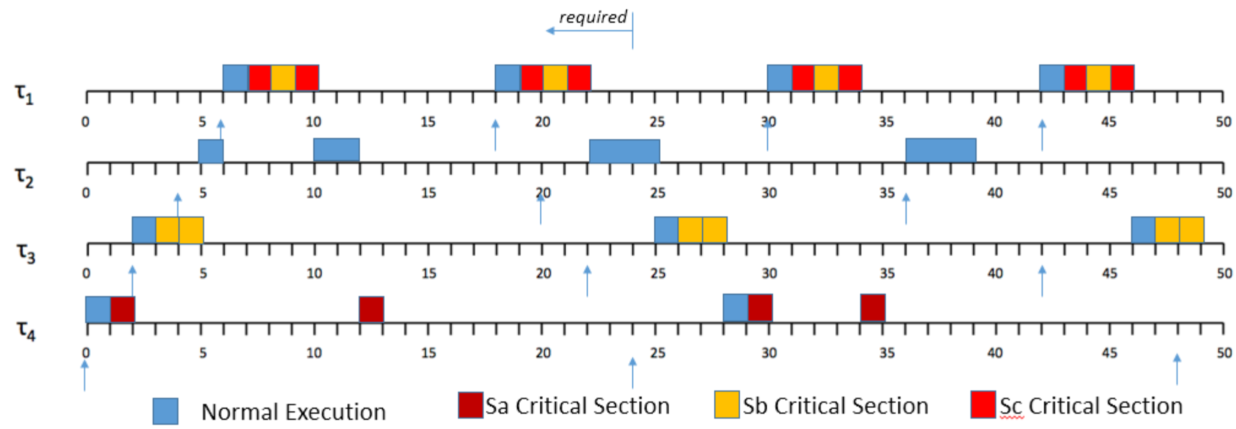
Priority ceiling(Sa) = priority(τ_4)
Priority ceiling(Sb) = priority(τ_1)
Priority ceiling(Sc) = priority(τ_1)



Note: Task τ_1 is blocked by τ_3 when trying to lock Sc, Task τ_3 which had locked Sb executes at τ_1's priority at time 7 until Sb is released at time 8.

b) Draw the timeline under RMS using Highest Locker Priority. Please mark all the critical sections.

Priority ceiling(Sa) = priority(τ_4) → also the highest locker priority
Priority ceiling(Sb) = priority(τ_1) → also the highest locker priority
Priority ceiling(Sc) = priority(τ_1) → also the highest locker priority



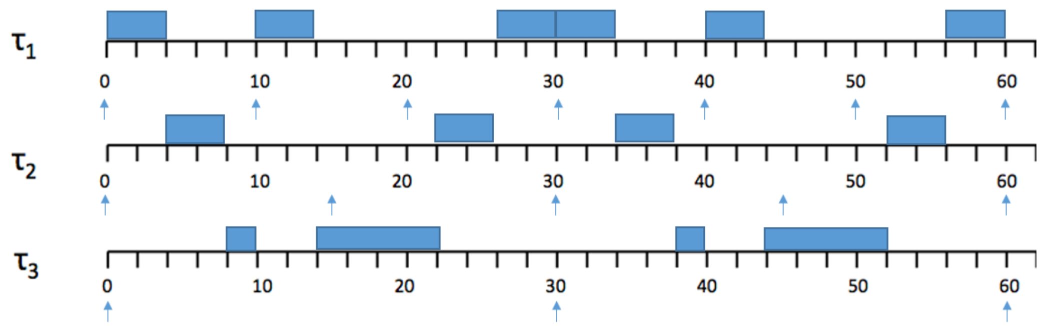
Note: Task τ_3 executes at the priority of τ_1 as soon as it locks Sb and therefore cannot be preempted by τ_1 or τ_2 until Sb is released at time 5.

Question 4. An overloaded question (20 points)

There are 3 real-time tasks in the system. Each task τ_i is characterized by $\{C_i,T_i\}$, its worst-case execution time and period parameters. The relative deadline of each task is the same as its period.

$\tau_1 = \{4,10\}$ → $U_1 = 0.4$
$\tau_2 = \{4,15\}$ → $U_2 = 0.2667$
$\tau_3 = \{10,30\}$ → $U_3 = 0.3333$ → $U_{total} = 0.4 + 0.2667 + 0.3333 = 1.0$

a) Determine if this given task set is schedulable or not using the Earliest Deadline First scheduling algorithm. Fill in the timeline to verify your answer. (Draw the time period from 0-60)



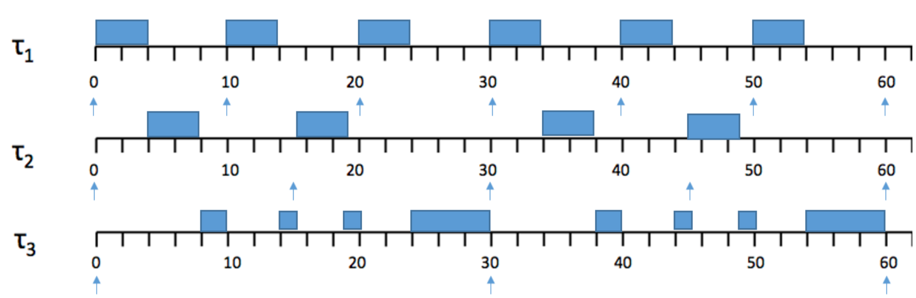
b) Determine if this given task set is schedulable or not using the Rate-Monotonic scheduling algorithm.

Note: The taskset is *not* harmonic since $T_1 = 10$ and $T_2 = 15$ are not integral multiples (or sub-multiples) of one another.
Since $U_{total} > U_{bound}(3) = 0.78$, we cannot use the schedulability bound test or the harmonic utilization bound to determine the schedulability of the entire taskset.
Since $U_1 + U_2 = 0.6667 < U_{bound}(2) = 0.828$, tasks τ_1 and τ_2 are schedulable.
For task τ_3, the exact completion time test must be run. We therefore have,

$a^3_0 = C_1 + C_2 + C_3 = 4 + 4 + 10 = 18$
$a^3_1 = C_3 + \text{ceiling}(18/10)*4 + \text{ceiling}(18/15)*4 = 10 + 2*4 + 2*4 = 26$
$a^3_2 = C_3 + \text{ceiling}(26/10)*4 + \text{ceiling}(26/15)*4 = 10 + 3*4 + 2*4 = 30$
$a^3_3 = C_3 + \text{ceiling}(30/10)*4 + \text{ceiling}(30/15)*4 = 10 + 3*4 + 2*4 = 30$

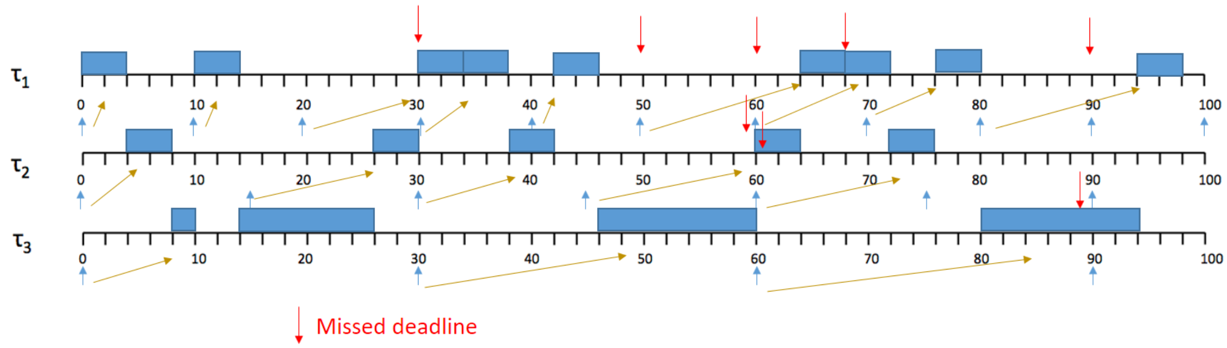
Since $a^3_2 = a^3_3$, we can stop. The completion time of τ_3 is 30 <= its deadline of 30. Hence, task τ_3 and the entire taskset is schedulable!

<u>*Comment*</u>: The timeline looks as follows.



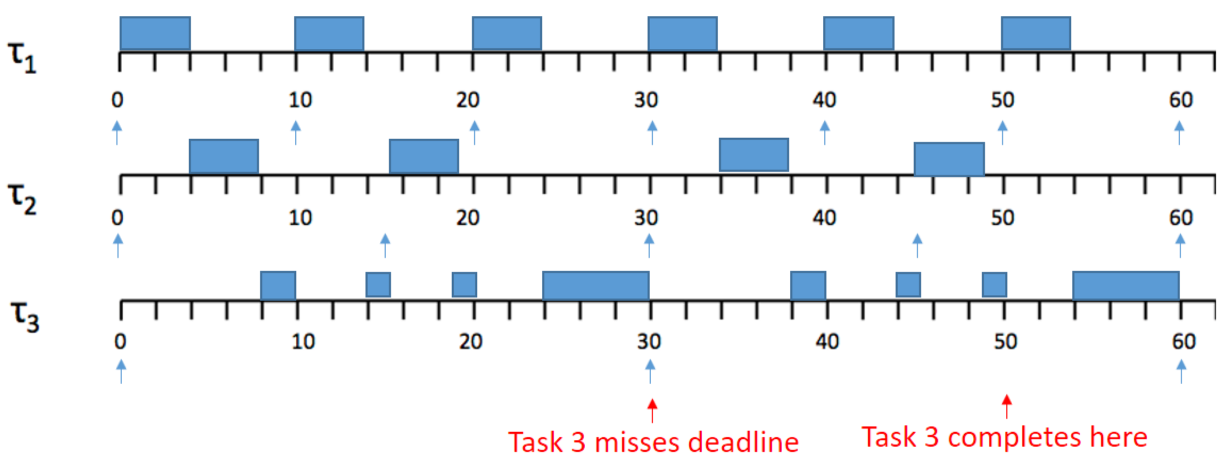
Now assume τ_3's worst-case execution time increases from 10 to 14.

c) Determine if the new task set is schedulable using the Earliest Deadline First scheduling algorithm. If the task set cannot be scheduled, indicate which tasks will miss their deadlines. Draw the timeline for this given task set to verify your answer.(Draw the time period from 0-90). If there is a tie, break when applicable in favor of the task that arrived earlier (else break the tie arbitrarily). If a task misses a deadline, it continues to be eligible to execute at the priority of its absolute deadline.



Note: <u>All</u> tasks miss their deadlines and the overloaded situation becomes worse over time. In other words, dynamic priority scheduling algorithms are 'unstable' under overloaded conditions.

d) Determine if the new task set from (c) is schedulable using the Rate-Monotonic scheduling algorithm. If the task set cannot be scheduled, figure out which task will miss the deadline. Draw the timeline for this given task set to verify your answer. (Draw the time period from 0-90)



Note: The lowest priority task $\boldsymbol{\tau_3}$ will miss its deadlines but tasks $\boldsymbol{\tau_1}$1 and $\boldsymbol{\tau_2}$ will meet all their future deadlines. This is referred to as the stability of RMS (fixed-priority scheduling) under overloaded conditions.

Question 5. Time to block. (12 points)

A real-time task set has 3 tasks τ_1, τ_2 and τ_3 in decreasing order of (fixed) priority. They share some logical resources like critical sections and global variables that are protected using mutexes.

- Task τ_1 accesses a mutex M_1 for 5 *ms.*
- Task τ_2 accesses two mutexes M_2 and M_3 for 7 *ms* and 5 *ms* respectively.
- Task τ_3 accesses three mutexes M_1, M_2 and M_4 for 8 *ms*, 12 *ms* and 12 *ms* respectively.

Note: There are no nested mutex accesses of any kind: one mutex is locked and then released, *before* another mutex is locked.

Please fill in the table below.

Blocking Term	Under Basic PIP	Under PCP	Under HLP	Under NPP
B_1	8	8	8	12
B_2	12	12	12	12
B_3	0	0	0	0

Note: The lowest-priority task experiences 0 blocking time since there is no lower-priority task to cause priority inversion under all these protocols (unless the OS refuses to schedule it even when the processor is idle – which would be a ridiculous OS and scheduling policy).

First, compute the priority ceilings of the mutexes.

Priority ceiling of M_1 = priority of τ_1.
Priority ceiling of M_2 = priority of τ_2
Priority ceiling of M_3 = priority of τ_2
Priority ceiling of M_4 = priority of τ_3

Under Basic PIP: A task τ can encounter priority inversion from one or more lower-priority tasks that use mutexes with priority ceilings greater than or equal to the priority of task τ. But each mutex can cause at most one critical section of priority inversion, and one task can also cause at most one priority inversion. Therefore,

- task τ_1 can only be blocked by the critical section of task τ_3 locking M_1. Hence, $B_1 = 8$.
- task τ_2 can be blocked by the critical sections of task τ_3 locking M_1 and M_2. However, both of of them cannot cause priority inversion – only one can (because when task τ_2 arrives, task τ_3 could have locked either M_1 or M_2 but not both). So, we pick *max(8, 12).*

Under PCP: A task τ can encounter priority inversion from no more than one lower-priority task that uses mutexes with priority ceilings greater than or equal to the priority of task τ. Therefore,

- task τ_1 can only be blocked by the critical section of task τ_3 locking M_1. Hence, $B_1 = 8$.
- task τ_2 can be blocked by one of the critical sections of task τ_3 locking M_1 or M_2. So, we pick *max(8, 12).*

Under HLP: Same as PCP when tasks do not suspend voluntarily within critical sections. That is, a task τ can encounter priority inversion from no more than one lower-priority task that uses mutexes with priority ceilings greater than or equal to the priority of task τ. The blocking terms are the same as PCP. (This is why HLP is a very good approximation of PCP. Their property differences come in only when tasks can suspend voluntarily inside a critical section – that's bad practice anyway).

Under NPP: Under the non-preemption protocol, a task starts executing at the highest system priority as soon as it locks a mutex. In other words, it acts like the HLP except that all mutexes are assigned a priority ceiling = the highest priority in the system. Hence, a task τ can encounter priority inversion from no more than one lower-priority task, but any task that locks a mutex can cause priority inversion. Hence,

$B_1 = \max(7, 5, 8, 12, 12) = 12$

$B_2 = \max(8, 12, 12) = 12$