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

# 18-648: Embedded Real-Time Systems

## Quiz #1

**Fall 2017**

**60 minutes**

**Instructions**

1. This is a **CLOSED-BOOK/CLOSED-NOTES quiz**.
2. Show all relevant work.
3. Partial credit may be given for some questions.
4. The use of a calculator is allowed.
5. The time limit will be *strictly* enforced.
6. Watch the screen for any clarifications.

---

**For Graders' Use Only**

Bonus: __________ / 1

1. __________ / 15

2. __________ / 4

3. __________ / 19

4. __________ / 12

**TOTAL. __________ / 50**

**Question 1. Fast Food.**

**(a) True/False Questions (12 points)**

i. FALSE A hard real-time system can only use hardware, while a soft real-time system can also use software.

ii. TRUE Given a periodic taskset, if any fixed-priority preemptive scheduling policy can render that taskset schedulable, then this taskset is also schedulable using the rate-monotonic scheduling policy.

iii. TRUE Under rate-monotonic scheduling, when the relative deadline is at the end of a task period, the schedulability of a task can be tested by checking its first deadline when all tasks arrive together.

iv. FALSE In a real-time system, processing must be done as fast as possible.

v. TRUE If every thread, that holds two mutexes *m1* and *m2* simultaneously, locks them in the same order, the system will never deadlock.

vi. TRUE Threads of the same process share their data and code segments.

vii. TRUE Using the Priority Ceiling Protocol (PCP) will bound the blocking time due to priority inversion.

viii. TRUE Only a limited number of threads can be created within a process.

ix. 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.

x. FALSE Fixed-priority preemptive schedulers have a higher schedulable utilization bound than dynamic-priority preemptive schedulers.

xi. FALSE Earliest deadline first (EDF) is a fixed-priority preemptive-scheduling scheme.

xii. TRUE Consider a fixed-priority scheduler which assigns random (fixed) priority values to periodic real-time tasks. There will be *never* be any priority assignments on some task sets for which this scheduler would find a feasible schedule but the rate-monotonic scheduler will not.

**(b) Circle one or more answers for the questions below. (3 points)**

I. User-space applications access kernel functions through:

a. A shell.
b. System calls.
c. Shared memory.
d. All of the above.
e. None of the above.

II. Which of the following protocols disallow a mutex *M* from being locked even if *M* is currently unlocked?

a. Basic Priority Inheritance Protocol (PIP)
b. Highest Locker Priority Protocol (HLP)
c. Priority Ceiling Protocol (PCP)
d. Non-Preemption Protocol (NPP)

III. The schedulability of a periodic taskset using rate-monotonic scheduling can potentially be determined using

a. A utilization bound test
b. A harmonicity test
c. Completion-time test

**Question 2. At a fork? Take it. (4 points)**
Given the following code, what is/are the possible output(s)? Assume that fork() succeeds.

```
count = 2;
if (pid = fork()) {
   count += 6;
   printf("%d\n", count);
   count += 1;
   return 0;
}

count += 2;
printf("%d\n", count);
return 0;
```

Possible Output #1:
8
4

Possible Output #2:
4
8

*Note*: In reality, printf() itself is not atomic and one could potentially also have outputs like “84\n\n” and “48\n\n”.

**Question 3. Schedules, schedules (19 points)**
Consider a periodic taskset with 3 tasks. Each task $\tau_i$ is characterized by $\{C_i, T_i\}$, its worst-case execution time and period parameters respectively. The relative deadline of each task is the same as its period.

$\tau_1 = \{7, 10\}$
$\tau_2 = \{3, 15\}$
$\tau_3 = \{3, 30\}$

**(a)** Determine whether the above taskset is schedulable using Earliest Deadline First scheduling algorithm. **(3 points)**

$U_{total} = (7/10) + (3/15) + (3/30) = 0.7 + 0.2 + 0.1 = 1.0$
Since $U_{total} \leq 1.0$, the taskset is schedulable under the EDF policy.

(b) Determine whether the above taskset is schedulable using the Rate-Monotonic Scheduling policy. 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 $a^i_{k+1} = a^i_k$ **(9 points)**

Applying the **utilization test**,
For $\tau_1$ alone, $U_1 = 0.7 < U_b(1) = 1.0$ so $\tau_1$ is schedulable.
For $\tau_2$, $U_1+U_2 = 0.7 + 0.2 = 0.9 > U_b(2) = 0.828$ – so test fails. Perform the Response Time (RT) test to check if $\tau_2$ is schedulable.

**RT test for $\tau_2$ ($i$ =2)**

$A^2_0 = C_1 + C_2 = 7 + 3 = 10$
$A^2_1 = C_2 + 7*\text{ceiling}(10/10) = 3 + 7*1 = 10$

Since $A^2_0 = A^2_1 = 10$, the RT test converges at time $t = 10 < T_2$ (=15), so $\tau_2$ is schedulable.

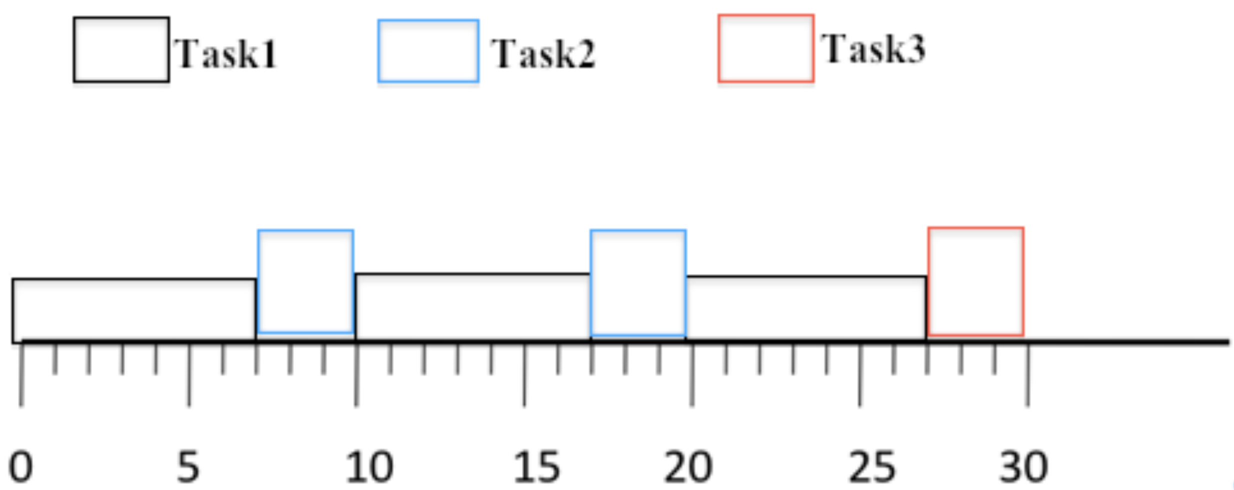
For $\tau_3$, $U_1+U_2+U_3 = 1.0 > U_b = 0.78$ fails. Perform RT test to check if schedulable.

**RT test for $\tau_3$ ($i$ =3)**

$A^3_0 = C_1 + C_2 + C_3 = 7 + 3 + 3 = 13$
$A^3_1 = C_3 + 7*\text{ceiling}(13/10) + 3*\text{ceiling}(13/15) = 3 + 7*2 + 3 * 1 = 20$
$A^3_2 = C_3 + 7*\text{ceiling}(20/10) + 3*\text{ceiling}(20/15) = 3 + 7*2 + 3 * 2 = 23$
$A^3_3 = C_3 + 7*\text{ceiling}(23/10) + 3*\text{ceiling}(23/15) = 3 + 7*3 + 3 * 2 = 30$
$A^3_4 = C_3 + 7*\text{ceiling}(30/10) + 3*\text{ceiling}(30/15) = 3 + 7*3 + 3 * 2 = 30$

Since $A^3_3 = A^3_4 = 30$, test converges at time $t$=30 ≤ $T_3$ (=30), and hence $\tau_3$ is schedulable.

(c) Validate your result of (b) by drawing the critical zone. **(5 points)**




- The longest response time for task $\tau_1$ is 7 (< deadline/period of 10).
- The longest response time for task $\tau_2$ is 10 (< deadline/period of 15)
- The longest response time for task $\tau_3$ is 30 (<= deadline/period of 30).

These are consistent with our response time tests as well.

Note: The RT test mathematically computes the above manual sequence of execution to determine the longest response time for a task.

(d) Is the taskset harmonic? **(2 points)**

No. 10 and 30 are harmonic, as are 15 and 30. But 10 and 15 are not harmonic with respect to each other.

*Note*: The taskset has 100% utilization but is not harmonic. However, RMS is still able to schedule this taskset at 100% utilization. All harmonic tasksets will yield 100% schedulable utilization under RMS, but not all non-harmonic tasksets can do the same. The least upper utilization bound of ln 2 is an example of the latter. However, the fact that some non-harmonic tasksets can reach 100% schedulable utilization for the simple RMS policy testifies to its practical effectiveness.

## Question 4. Time to Block. (12 points)

A real-time taskset has 4 tasks $\tau_1$, $\tau_2$, $\tau_3$ and $\tau_4$ in **decreasing** order of (fixed) priority. They share some logical resources like critical sections and global variables that are protected using mutexes.

- Task $\tau_1$ accesses a mutex $M_1$ for 5 *ms.*
- Task $\tau_2$ accesses two mutexes $M_2$ and $M_3$ in non-nested fashion for 8 *ms* and 6 *ms* respectively.
- Task $\tau_3$ accesses three mutexes $M_1$, $M_2$ and $M_4$ in non-nested fashion for 10 *ms*, 12 *ms* and 14 *ms* respectively.
- Task $\tau_4$ accesses mutexes $M_3$ and $M_4$ in non-nested fashion for 13 *ms* and 15 *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 BIP | Under PCP | Under HLP | Under NPP |
|---|---|---|---|---|
| $B_1$ | 10 | 10 | 10 | max(8,6,10,12,14,13,15) = 15 |
| $B_2$ | max(10,12) + 13 = 25 | max(10,12,13) = 13 | max(10,12,13) = 13 | max(10,12,14,13,15) = 15 |
| $B_3$ | max(13,15)=15 | max(13,15)=15 | max(13,15)=15 | max(13,15)=15 |
| $B_4$ | 0 | 0 | 0 | 0 |

Show work below.

**Steps:**

1. Determine the priority ceiling for each mutex.
2. Under each protocol, for each task, determine the set of lower-priority critical sections that can cause priority inversion to the task: this set consists of the critical sections guarded by mutexes with a priority ceiling greater than or equal to the task priority.
3. Then, apply the properties of the protocol to determine the worst-case blocking time of a task.

**Priority Ceilings of Mutexes:**

Priority ceiling of $M_1$ = priority of $\tau_1$

Priority ceiling of $M_2$ = priority of $\tau_2$

Priority ceiling of $M_3$ = priority of $\tau_2$

Priority ceiling of $M_4$ = priority of $\tau_3$

**Under PIP:**

For $\tau_1$, the set of lower-priority critical sections to consider: the $M_1$ critical section of task $\tau_3$

For $\tau_2$, the set of lower-priority critical sections to consider: the critical sections of tasks $\tau_3$ and $\tau_4$ that access $M_1$, $M_2$ and $M_3$. Each lower-priority task can contribute only one (outermost) critical section and each mutex can contribute only one critical section. Hence, we have $B_2$ = max(10,12) from task 2 + 13 from task 3.

For $\tau_3$, the set of lower-priority critical sections to consider: both the critical sections of task $\tau_4$. But a lower-priority task can contribute only one (outermost) critical section. Hence, we have $B_3$ = max(13, 15).

**Under PCP and HLP**: (these have the same blocking times for tasks when tasks do not suspend within a critical section):

For $\tau_1$, the set of lower-priority critical sections to consider: the $M_1$ critical section of task $\tau_3$.

For $\tau_2$, the set of lower-priority critical sections to consider: the critical sections of tasks $\tau_3$ and $\tau_4$ that access $M_1$, $M_2$ and $M_3$. Only one of these critical sections can cause priority inversion. Hence, we have $B_2$ = max(10,12,13).

For $\tau_3$, the set of lower-priority critical sections to consider: both the critical sections of task $\tau_4$. Only one of these critical sections can cause priority inversion. Hence, we have $B_3$ = max(13, 15).

**For NPP:**

Under NPP, any lower-priority critical section can cause priority inversion but at most one can. Hence, $B_i$ is given by the longest critical section of all its lower-priority tasks.

Another equivalent alternative is to consider the priority ceilings of all mutexes to be the highest priority in the system. Hence, all lower-priority critical sections are eligible to cause priority inversion to a task but only one of them can. For the worst case, we pick the longest.

**Question 4. Time to Block. (12 points) - original question on paper without fix during exam**

A real-time taskset has 4 tasks $\tau_1$, $\tau_2$, $\tau_3$ and $\tau_4$ in **increasing** order of (fixed) priority. They share some logical resources like critical sections and global variables that are protected using mutexes.

- Task $\tau_1$ accesses a mutex $M_1$ for 5 *ms.*
- Task $\tau_2$ accesses two mutexes $M_2$ and $M_3$ in non-nested fashion for 8 *ms* and 6 *ms* respectively.
- Task $\tau_3$ accesses three mutexes $M_1$, $M_2$ and $M_4$ in non-nested fashion for 10 *ms*, 12 *ms* and 14 *ms* respectively.
- Task $\tau_4$ accesses mutexes $M_3$ and $M_4$ in non-nested fashion for 13 *ms* and 15 *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 BIP | Under PCP | Under HLP | Under NPP |
|---|---|---|---|---|
| $B_1$ | 0 | 0 | 0 | 0 |
| $B_2$ | 5 | 5 | 5 | 5 |
| $B_3$ | max(8,6) + 5 =13 | max(5,8,6)=8 | max(5,8,6)=8 | max(5,8,6)=8 |
| $B_4$ | 14 + 6 = 20 | max(14,6) = 14 | max(14,6) = 14 | max(5,8,6,10,12,14) = 14 |

Show work below.

**Steps:**

1. Determine the priority ceiling for each mutex.
2. Under each protocol, for each task, determine the set of lower-priority critical sections that can cause priority inversion to the task: this set consists of the critical sections guarded by mutexes with a priority ceiling greater than or equal to the task priority.

3. Then, apply the properties of the protocol to determine the worst-case blocking time of a task.

**Priority Ceilings of Mutexes:**

Priority ceiling of $M_1$ = priority of $\tau_3$

Priority ceiling of $M_2$ = priority of $\tau_3$

Priority ceiling of $M_3$ = priority of $\tau_4$

Priority ceiling of $M_4$ = priority of $\tau_4$

**Under PIP:**

For $\tau_4$, the set of lower-priority critical sections to consider: the $M_3$ critical section of task $\tau_2$ and the $M_4$ critical section of task $\tau_3$

For $\tau_3$, the set of lower-priority critical sections to consider: the critical sections of tasks $\tau_1$ and $\tau_2$ that access $M_1$ and $M_2$. Each lower-priority task can contribute only one (outermost) critical section and each mutex can contribute only one critical section. Hence, we have $B_3$ = max(8,6) from $\tau_2$ + 5 from $\tau_1$.

For $\tau_2$, the set of lower-priority critical sections to consider: the critical section of $M_1$ task $\tau_1$.

**Under PCP and HLP**: (these have the same blocking times for tasks when tasks do not suspend within a critical section):

For $\tau_4$, the set of lower-priority critical sections to consider: the $M_3$ critical section of task $\tau_2$ and the $M_4$ critical section of task $\tau_3$ = max(6, 14)

For $\tau_3$, the set of lower-priority critical sections to consider: the critical sections of tasks $\tau_1$ and $\tau_2$ that access $M_1$, $M_2$ and $M_3$. Only one of these critical sections can cause priority inversion. Hence, we have $B_3$ = max(5,8,6).

For $\tau_2$, the set of lower-priority critical sections to consider: the critical section of $M_1$ task $\tau_1$.

**For NPP:**

Under NPP, any lower-priority critical section can cause priority inversion but at most one can. Hence, $B_i$ is given by the longest critical section of all its lower-priority tasks.

Another equivalent alternative is to consider the priority ceilings of all mutexes to be the highest priority in the system. Hence, all lower-priority critical sections are eligible to cause priority inversion to a task but only one of them can. For the worst case, we pick the longest.