Real-Time Systems

Solutions to final exam March 14, 2016

PROBLEM 1

a) True: TinyTimber allows multiple methods to execute concurrently, given that each method reside is a separate object.

b) False: The behavior of a data cache is in general much more difficult to predict.

c) False: The response-time test for global fixed-priority scheduling is only a sufficient test.

d) False: By deadline inversion me mean a situation in EDF scheduling where an urgent task is being blocked by a less urgent task (due to e.g. sharing the same resource) that has a deadline later in time.

e) True: ICPP cannot cause deadlock if all task use the protocol to access shared resources.

f) True: For a sufficient feasibility test a positive answer guarantees that the task set is schedulable. Therefore, if the task set is not schedulable the answer from the test must have been a "no".

PROBLEM 2

a) CAN is contention-based (collision-based).

b) See lecture notes for Lecture 7 (slide 31).

c) See lecture notes for Lecture 7 (slide 32).

PROBLEM 3

a) The WCET of main is dependent on the WCET of functions "FuncA" and "FuncB".

WCET of "main":

$$
\begin{aligned}
&WCET(main) = \{Dec, a\} + \{Dec, b\} + \{Dec, c\} + \{Dec, d\} + \{Dec, e\} + \{Dec, result\} \\
&+ \{Dec, threshold\} + \{Assign, threshold = 100\} + \{Assign, result = -1\} + \{Assign, a = 3\} \\
&+ \{Assign, b = 4\} + \{abs, abs(a)\} + \{abs, abs(b)\} + \{call, FuncA(abs(b), abs(a))\} \\
&+ WCET(FuncA(abs(4), abs(3))) + \{Assign, c = FuncA(abs(b), abs(a))\} + \{comp, c > threshold\} \\
&+ \{abs, abs(a)\} + \{abs, abs(b)\} + \{call, FuncA(abs(a), abs(b))\} + WCET(FuncA(abs(3), abs(4))) \\
&+ \{Assign, d = FuncA(abs(a), abs(b))\} + \{comp, d > threshold\} + \{add, c + d\} + \{Assgn, e = c + d\} \\
&+ \{comp, e > threshold\} + \{call, FuncB(e)\} + \{Assign, result = FuncB(e)\} + WCET(FuncB(3)) \\
&= 50 + WCET(FuncA(4, 3)) + WCET(FuncA(3, 4)) + WCET(FuncB(145))
\end{aligned}
$$

WCET of "FuncA": There are three cases for calculating the WCET of FuncA:

$Case(i) y == 0, Case(ii) y = 1, Case(iii) y > 1.$

$Case(i) WCET(FuncA(x,y)) = \{Dec, z\} + \{Assign, z = 0\} + \{Comp, y == 0\} + \{Assign, z = 1\}$
$+ \{return, z\} = 7$
$Case(ii) WCET(FuncA(x,y)) = \{Dec, z\} + \{Assign, z = 0\} + \{Comp, y == 0\} + \{Comp, y == 1\}$
$\{Assign, z = x\} + \{return, z\} = 9$
$Case(iii) WCET(FuncA(x,y)) = \{Dec, z\} + \{Assign, z = 0\} + \{Comp, y == 0\} + \{Comp, y == 1\}$
$+ \{sub, y-1\} + \{call, FuncA(x, y-1)\} + WCET(FuncA(x, y-1)) + \{mul, x * FuncA(x, y-1)\}$
$+ \{Assign, z = x * FuncA(x, y-1)\} + \{return, z\} = 18 + WCET(FuncA(x, y-1))$
$WCET(FuncA(4,3)) = 18 + WCET(FuncA(4,2)) = 18 + 18 + WCET(FuncA(4,1)) =$
$18 + 18 + 9 = 45$
$WCET(FuncA(3,4)) = 18 + WCET(FuncA(3,3)) = 18 + 18 + WCET(FuncA(3,2)) =$
$18 + 18 + 18 + WCET(FuncA(3,1)) = 18 + 18 + 18 + 9 = 63$

WCET of "FuncB":

$WCET(FuncB(x)) = \{Dec, y\} + \{Assign, y = x\} + \{mod, y\%3\} + \{Comp, (y\%3)! = 0\}$
$\{mod, y\%3\} + \{mod, ((y\%3)\%2)\} + \{Comp, ((y\%3)\%2)\} + \{sub, y-1\} + \{return, y-1\}$
$= WCET(FuncB(145)) = 26$

WCET of "main":

$$WCET(main) = 50 + 45 + 63 + 26 = 184$$

The deadline is met

b) There are seven addition/subtraction operations. The cost of each of them cannot be more than 4 μs so that the deadline is met.

c) The WCET estimates should be pessimistic to make sure assumptions made in the schedulability analysis of hard real-time tasks also apply at run time. The estimates should be tight to avoid unnecessary waste of resources during scheduling of hard real-time tasks.

PROBLEM 4

a) The tasks T1, T2 and T3 should normally reside in three separate objects, but since their execution is precedence-constrained a solution where they share one object (A) is also correct. Task BG needs to reside in an object that is separate (object B) from the hard-real-time tasks.

```
void T1(TaskObj *self, int u) {

    Action300(); // Do work for 300 microseconds

    BEFORE(USEC(1200), self, T2, 0); // Keep current baseline
}

void T2(TaskObj *self, int u) {

    Action800(); // Do work for 800 microseconds

    BEFORE(USEC(2100), self, T3, 0); // Keep current baseline
}

void T3(TaskObj *self, int u) {

    Action500(); // Do work for 500 microseconds

    SEND(USEC(2400), USEC(1600), self, T1, 0);
}

void BG(TaskObj *self, int u) {

    Load700(); // Do background work for 700 microseconds

    SEND(USEC(1800), USEC(1800), self, BG, 0);
}

void kickoff(TaskObj *self, int u) {

    BEFORE(USEC(1600), &A, T1, 0);
    BEFORE(USEC(1800), &B, BG, 0);
}

main() {
    return TINYTIMBER(&A, kickoff, 0);
}
```

b) By simulating the execution of the tasks, assuming an EDF scheduler, we can see that task T3 will miss its deadline at t=21 (having two more time units to execute). Conclusion: It is not possible to guarantee that task T3 will meet its deadline.

PROBLEM 5

We start by observing that task τ_1 has a first arrival time that differs from that of the other tasks. This means that the use of a utilization-based or response-time-based schedulability test may become overly pessimistic IF there exists no point in time in the schedule where all tasks arrive at the same time. This, in turn, could mean that, should the test fail, the task set could potentially still be schedulable.

Luckily, by observing the given periods and offsets, we can see that there does exist a point in time where all tasks arrive at the same time, namely at $t = 3T$ (where the arrivals of the 4th instance of τ_1, the 4th instance of τ_2 and the 3rd instance of τ_3 coincide). We can then use this as the critical instant in our analysis.

Our first candidate method for schedulability analysis is Liu and Layland's classic utilization-based test. For three tasks, the schedulability bound is $U_{lub} = 3(2^{1/3} - 1) \approx 0.780$. Unfortunately, the accumulated task utilization, $U = 1/4 + 9/20 + 2/15 \approx 0.833$, exceeds the guarantee bound, and the test does not provide any useful information.

We must, consequently, resort to response-time analysis. Since RM is used, the task priorities are determined by the task periods. To that end, task τ_1 has highest priority (shortest period) och process τ_3 has lowest priority.

First, we derive the execution time of each task:

$C_1 = U_1 \cdot T_1 = 1/4 \cdot 0.8T = 0.2T$

$C_2 = U_2 \cdot T_2 = 9/20 \cdot T = 0.45T$

$C_3 = U_3 \cdot T_3 = 2/15 \cdot 1.5T = 0.2T$

We then calculate the response time of each task and compare it against the corresponding task deadline:

$R_1 = C_1 = 0.2T < D_1 = 0.8T.$

$R_2 = C_2 + \lceil \frac{R_2}{T_1} \rceil \cdot C_1$. Assume that $R_2^0 = C_2 = 0.45T$:

$R_2^1 = 0.45T + \lceil \frac{0.45T}{0.8T} \rceil \cdot 0.2T = 0.45T + 1 \cdot 0.2T = 0.65T$

$R_2^2 = 0.45T + \lceil \frac{0.65T}{0.8T} \rceil \cdot 0.2T = 0.45T + 1 \cdot 0.2T = 0.65T < D_2 = T$

$R_3 = C_3 + \lceil \frac{R_3}{T_1} \rceil \cdot C_1 + \lceil \frac{R_3}{T_2} \rceil \cdot C_2$. Assume that $R_3^0 = C_3 = 0.2T$:

$R_3^1 = 0.2T + 0 + \lceil \frac{0.2T}{0.8T} \rceil \cdot 0.2T + \lceil \frac{0.2T}{T} \rceil \cdot 0.45T = 0.2T + 1 \cdot 0.2T + 1 \cdot 0.45T = 0.85T$

$R_3^2 = 0.2T + 0 + \lceil \frac{0.85T}{0.8T} \rceil \cdot 0.2T + \lceil \frac{0.85T}{T} \rceil \cdot 0.45T = 0.2T + 2 \cdot 0.2T + 1 \cdot 0.45T = 1.05T$

$R_3^3 = 0.2T + 0 + \lceil \frac{1.05T}{0.8T} \rceil \cdot 0.2T + \lceil \frac{1.05T}{T} \rceil \cdot 0.45T = 0.2T + 2 \cdot 0.2T + 2 \cdot 0.45T = 1.5T$

$R_3^4 = 0.2T + 0 + \lceil \frac{1.5T}{0.8T} \rceil \cdot 0.2T + \lceil \frac{1.5T}{T} \rceil \cdot 0.45T = 0.2T + 2 \cdot 0.2T + 2 \cdot 0.45T = 1.5T \leq D_3 = 1.5T$

Conclusion: all tasks meet their deadlines!

PROBLEM 6

Define priority according to the following: H = highest priority, M = medium priority and L = lowest priority. Since DM is used, the task priorities are determined by the task deadlines. To that end, task τ_1 has highest priority (shortest deadline) och process τ_3 has lowest priority.

We first determine the ceiling priority of each resource:

$\text{ceil}\{R_a\} = \max\{H, M, L\} = H$ (since τ_1, τ_2 and τ_3 can request the semaphore)

$\text{ceil}\{R_b\} = \max\{M, L\} = M$ (since τ_2 and τ_3 can request the semaphore)

$\text{ceil}\{R_c\} = \max\{H, M, L\} = H$ (since τ_1, τ_2 and τ_3 can request the semaphore)

We then identify, for each task τ_i, which tasks with lower priority that can block τ_i and thereby determines the corresponding blocking factor B_i. Note that nested blocking is used by all tasks. This could lead to accumulated critical region blocking times in the final blocking factor.

$B_1 = \max\{H_{2,a}, H_{2,c} + H_{2,b}, H_{3,a}, H_{3,c}\} = \max\{1, 4, 2, 2\} = 4$ (since τ_1 can be blocked by τ_2 och τ_3 that locks resources whose ceiling priorities are higher than, or equal to, the priority of τ_1; note accumulated blocking time due to the nested blocking by τ_2)

$B_2 = \max\{H_{3,c}, H_{3,b} + H_{3,a}\} = \max\{2, 5\} = 5$ (since τ_2 can be blocked by τ_3 that locks resources whose ceiling priorities are higher than, or equal to, the priority of τ_2; note accumulated blocking time due to the nested blocking by τ_3)

$B_3 = 0$ (since τ_3 has the lowest priority among all tasks, and thereby per definition cannot suffer blocking)

We now calculate the response time of each task and compare it against the corresponding task deadline:

$R_1 = C_1 + B_1 = 6 + 4 = 10 \leq D_1 = 10.$

$R_2 = C_2 + B_2 + \lceil \frac{R_2}{T_1} \rceil \cdot C_1$. Assume that $R_2^0 = C_2 = 7$:

$R_2^1 = 7 + 5 + \lceil \frac{7}{31} \rceil \cdot 6 = 7 + 5 + 1 \cdot 6 = 18.$

$R_2^2 = 7 + 5 + \lceil \frac{18}{31} \rceil \cdot 6 = 7 + 5 + 1 \cdot 6 = 18 > D_2 = 17.$

$R_3 = C_3 + B_3 + \lceil \frac{R_3}{T_1} \rceil \cdot C_1 + \lceil \frac{R_3}{T_2} \rceil \cdot C_2$. Assume that $R_3^0 = C_3 = 10$:

$R_3^1 = 10 + 0 + \lceil \frac{10}{31} \rceil \cdot 6 + \lceil \frac{10}{29} \rceil \cdot 7 = 10 + 0 + 1 \cdot 6 + 1 \cdot 7 = 23$

$R_3^2 = 10 + 0 + \lceil \frac{23}{31} \rceil \cdot 6 + \lceil \frac{23}{29} \rceil \cdot 7 = 10 + 0 + 1 \cdot 6 + 1 \cdot 7 = 23 < D_3 = 25$

Conclusion: task τ_2 may miss its deadline!

PROBLEM 7

a) Advantages with global scheduling:

- Supported by most multiprocessor operating systems (Windows 7, Mac OS X, Linux).
- Effective utilization of processing resources (unused processor time can easily be reclaimed, for example when a task does not execute its full WCET).

Disadvantages with global scheduling:

- Weak theoretical framework (few results from the uniprocessor case can be used).

b) The guarantee utilization bound for RM-US is $\frac{m^2}{3m-2}$. Since $m = 3$, we have $\frac{m^2}{3m-2} = 9/7 \approx 1.285$. Therefore, all deadlines will be met because total utilization 1.25 is not larger than 1.285.

c-i) Let the two processors are indexed as P1 and P2. Task τ_4 with the smallest period is assigned to processor P1. Since task τ_1 has utilization 0.6, it must be assigned to P2. After assigning τ_1 and τ_4, processor P1 and P2 have total utilization 0.5 and 0.6, respectively.

Consider task τ_2 that has period 20. If τ_2 is assigned to P1, then we must have $C_2/20 + 0.5 \leq 2(\sqrt{2}-1)$ according to Liu and Layland's sufficient test for one processor. This implies $C_2 \leq 6.5685$. Therefore, the largest integer value of C_2 is 6.

If $C_2 = 6$, then $C_3 = 2C_2 = 12$. The utilization of task τ_3 with $C_3 = 12$ is $12/40 = 0.3$. And, there is no processor where task τ_3 can be assigned.

If $C_2 = 5$, then $C_3 = 10$ and the utilization of τ_3 is $10/40 = 0.25$. And, there is no processor where task τ_3 can be assigned.

If $C_2 = 4$, then $C_3 = 8$ and the utilization of τ_3 is $8/40 = 0.20$. Now task τ_3 can be assigned to P2. **Therefore, the largest value of C_3 is 8.**

We have the following tasks-to-processors assignment:

$P1 \leftarrow \tau_2, \tau_4$ where $C_3 = 8$

$P2 \leftarrow \tau_1, \tau_3$

c-ii) The new task τ_5 can be assigned to P1 since total utilization of the tasks τ_4, τ_2 already assigned to P1 is 0.7. Task τ_5's utilization is 0.05 and can be assigned to P1.

$P1 \leftarrow \tau_2, \tau_4, \tau_5$ where $C_3 = 8$

$P2 \leftarrow \tau_1, \tau_3$
