Real-Time Systems

Solutions to final exam March 6, 2012

PROBLEM 1

a) False: The utilization guarantee bound for RM-US converges towards 33.3% as the number of processors become very large.

b) True: Too large over-estimation of the execution time of a task will leave less slack for the execution of other tasks in the schedulability analysis.

c) False: Sporadic tasks can be used in a real-time system with hard timing constraints since such tasks have a guaranteed minimum interarrival time, thus giving an upper bound of these tasks' utilization of the processor.

d) False: The critical instant refers to a point in time when the response time of an analyzed task is maximized. In single-processor system the critical instant occurs when the task arrives at the same time as all tasks with higher priority.

e) False: By following certain guidelines of how to request resources it is possible to avoid deadlock.

f) False: By deadline inversion we mean a situation where a task with high priority (deadline close in time) is blocked from executing because another task with lower priority (deadline later in time) is holding an exclusive resource needed by the high-priority task.

PROBLEM 2

a) Waiting for a corresponding time slot (TTP/C), Waiting for a transmission token (FDDI), Waiting for a contention-free transmission (Ethernet), Waiting for network priority negotiation (CAN).

b) The message transmission delay consists of:

Frame delay: $t_{frame} = N_{frame}/R$, where N_{frame} is the message length (in bits) and R is the data rate (in bits/s).

Propagation delay: $t_{prop} = L/v$, where L is the communication distance (in m) and v is the signal propagation velocity (in m/s).

c) Message queuing delay for Ethernet can in general not be bounded because a re-transmission will happen every time a collision on the communication medium occurs.

d) The binary countdown protocol:

1. Each node with a pending message waits until bus is idle.
2. The node begins transmitting the highest-priority message pending on the node. Identifier is transmitted first, in the order of most-significant bit to least-significant bit.
3. If a node transmits a recessive bit (1) but sees a dominant bit (0) on the bus, then it stops transmitting since it is not transmitting the highest-priority message in the system.
4. The node that transmits the last bit of its identifier without detecting a bus inconsistency has the highest priority and can start transmitting the body of the message.

PROBLEM 3

a) The WCET of main is dependent on the WCET of functions “methA”, “methB” and “multiply”. So, we first calculate the WCET of functions “methA”, “methB” and “multiply”.

WCET of “multiply”

WCET(multiply(a,b))= {multiplication}+{return}=5+2=7

WCET of “methA”

There are two cases for calculating the WCET of methA: case (i) b==0 and case(ii) b >0

Case(i)

WCET(methA(a,b==0))={dec, p}+{dec, i}+{assign, p}+{assign, i}+

{compare, b==0}+{return} =1+1+1+1+2+2=8

Case(ii)

The WCET of methA for this case depends on the number of times the while loop executes. The WCET of the while loop for parameter b is given as “WCET(whileLoop, b)”. Therefore, we have

WCET(methA(a,b>0))={dec, p}+{dec, i}+{assign, p}+{assign, i}+

{compare, b==0}+ WCET(whileLoop, b)+ {return}

=1+1+1+1+2+(16b-14)+2=16b-6

where **WCET(whileLoop, b)=(16b-14)**. The value of WCET(whileLoop, b) is calculated as follows. The body of the while loop executes $(b-1)$ times and the comparison statement of the while is checked b times.

WCET(whileLoop, b)= b * {comparison}+ (b-1)* ({call, multiply(p,a)}

+{WCET(multiply(p,a))}+{assign, p}+{add, i+1}+{assign, i})

=b * 2+(b-1)*(2+7+1+3+1)=16b-14

WCET of “methB”

There are two cases for calculating the WCET of methB: case (i) b==1 and case(ii) b >1

Case(i)

WCET(methB(a,b==1))={compare, b==1}+{return}=2+2=4

Case(ii)

WCET(methB(a,b>1))={compare, b==1}+{sub, b-1}+{call, methB(a,b-1)}
+WCET(methB(a,b-1))+{call, multiply(a,methB(a,b-1))}
+WCET(multiply(a,methB(a,b-1))+{return}
=2+3+2+WCET(methB(a,b-1))+2+7+2=18+WCET(methB(a,b-1))

WCET of "main"

Now we can calculate the WCET of main

WCET(main)={dec, ans}+{dec, x}+{dec, y}+{assign,x}+{assign,y}
+{call, methA(2,3)}+WCET(methA(2,3))+{call, methB(2,3)}
+WCET(methB(2,3))+ {compare,methA(2,3)>methB(2,3) }+{assign, ans}+return
=1+1+1+1+1+2+WCET(methA(2,3))+2+WCET(methB(2,3))+2+1+2
=14+WCET(methA(2,3))+WCET(methB(2,3))
=14+42+40=96 (deadline is missed!!)

where

WCET(methA(2,3))=16*3-6=42

and,

WCET(methB(2,3))=18+WCET(methB(2,2))=18+18+WCET(methB(2,1))=18+18+4=40

b) The false paths are:

In methA() the statement "return 1" is never executed since the value of b is not 0 since the value for formal parameter b in methA(a,b) is 3.

The condition "if (methA(x,y) > methB(x,y))" in the main() function is never true since both methA and methB compute *x raised to the power y*. Thus, the two statements "ans='T';" and "x=x+y;" are never executed.

PROBLEM 4

a), b) The final code should look similar to this:

```
#include TinyTimber.h

typedef struct {
    Object super;
    char *id;
} PeriodicTask;

Object app = initObject();
PeriodicTask ptask1 = { initObject(), "Task 1" };
PeriodicTask ptask2 = { initObject(), "Task 2" };
```

```
Time max_wcet = 0;

void T1(PeriodicTask *self, int u) {
    Time start;
    Time diff;

    start = CURRENT_OFFSET();

    Action1(); // procedure doing time-critical work

    diff = CURRENT_OFFSET() - start;
    if (diff > max_wcet)
        max_wcet = diff;

    SEND(MSEC(85), MSEC(30), self, T1, 0);
}

void T2(PeriodicTask *self, int u) {
    Action2(); // procedure doing time-critical work

    SEND(MSEC(105), MSEC(55), self, T2, 0);
}

void kickoff(PeriodicTask *self, int u) {
    SEND(MSEC(20), MSEC(30), &ptask1, T1, 0);
    SEND(MSEC(0), MSEC(55), &ptask2, T2, 0);
}

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

c) The priorities for the periodic activities are given by the deadlines in the SEND() calls, since the TinyTimber kernel uses earliest-deadline-first scheduling.

PROBLEM 5

a) **Priority Inheritance Protocol**: When a task τ_i blocks one or more higher-priority tasks, it temporarily assumes (inherits) the highest priority of the blocked tasks.

a) **Priority Ceiling Protocol**: Each resource is assigned a priority ceiling equal to the priority of the highest-priority task that can lock it. Then, a task τ_i is allowed to enter a critical section only if its priority is higher than all priority ceilings of the resources currently locked by tasks other than τ_i. When the task τ_i blocks one or more higher-priority tasks, it temporarily inherits the highest priority of the blocked tasks.

Advantages of PCP over PIP: (i) No deadlock: priority ceilings prevent deadlocks, (ii) No chained blocking: with PCP a task can be blocked at most the duration of one critical section (PIP can have chained blocking).

c) Define the priorities as follows: H = highest priority, M_H = second highest priority, M_L = third highest priority and L = lowest priority. Since rate-monotonic scheduling is used, the task priorities are as follows: $prio(\tau_1) = H$, $prio(\tau_2) = L$, $prio(\tau_3) = M_L$, and $prio(\tau_4) = M_H$.

We first determine the ceiling priority for each semaphore:

ceil$\{R_a\} = \max\{\text{H}, \text{L}\} = \text{H}$ (since τ_1 och τ_2 may lock the semaphore)

ceil$\{R_b\} = \max\{\text{M}_\text{H}, \text{M}_\text{L}\} = \text{M}_\text{H}$ (since τ_3 och τ_4 may lock the semaphore)

ceil$\{R_c\} = \max\{\text{H}, \text{M}_\text{H}\} = \text{H}$ (since τ_1 och τ_4 may lock the semaphore)

We then identify, for each task τ_i, what tasks with lower priority may block τ_i and thereby cause the corresponding blocking factor B_i:

$B_1 = \max\{H_{2,a}, H_{4,c}\} = \max\{2, 3\} = 3$ (since τ_1 may be blocked by τ_2 and τ_4 who lock semaphores whose ceiling priorities are higher than or equal to the priority of τ_1)

$B_2 = 0$ (since τ_2 has lowest priority of all tasks, and thereby per definition cannot be subject to blocking)

$B_3 = \max\{H_{2,a}\} = 2$ (since τ_3 may be blocked by τ_2 who locks a semaphore whose ceiling priority is higher than or equal to the priority of τ_3)

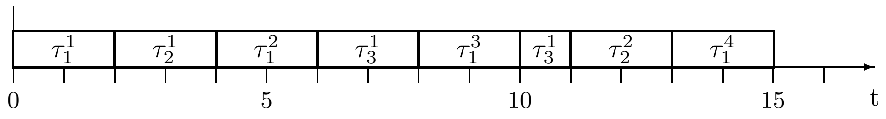
$B_4 = \max\{H_{2,a}, H_{3,b}\} = \max\{2, 1\} = 2$ (since τ_4 may be blocked by τ_2 and τ_3 who lock semaphores whose ceiling priorities are higher than or equal to the priority of τ_2)

PROBLEM 6

a) Time-table scheduling works as follows:

- Create a circular queue that corresponds to the time table: Each element in the queue contains start and finish times for a certain task (or task segment in case of preemptive scheduling). The elements in the queue are sorted by the start time
- Use clock interrupts: When a task starts executing, a real-time clock is programmed to generate an interrupt at the tasks expected finish time. When the interrupt occurs, the next task (the one whose start time is closest in time) in the circular queue is fetched and the system waits until that tasks given start time is due.

b) The length of the schedule is: $LCM(T_1, T_2, T_3) = LCM(4, 8, 16) = 16$. By simulating earliest-deadline-first scheduling we get the following schedule:

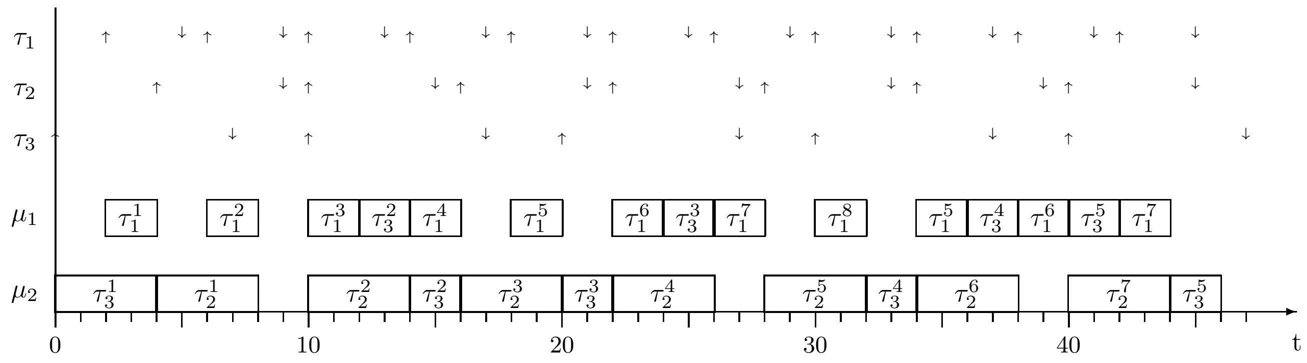


c) The generated schedule is the best possible (optimal). In a schedule generated by, for example, the deadline monotonic policy task τ_3 will miss its deadline.

PROBLEM 7

Since deadline-monotonic scheduling is used, the task priorities are as follows: $prio(\tau_1) = \text{H}$, $prio(\tau_2) = \text{M}$, $prio(\tau_3) = \text{L}$.

We generate a multiprocessor schedule with tasks τ_1 and τ_2 (having the highest priorities) running on one processor each. Task τ_3 is scheduled in the remaining time slots according to the following diagram (covering $t = 0$ to $t = 47$):



We observe that there is a critical instant at $t = 30$ in the diagram, maximizing the response time of τ_3 ($R_3 = 8$). Since $R_3 > D_3$ the schedule is not feasible.

Note that $t = 10$ is not a critical instant (which it would have been for a single-processor system)!