# Real-Time Systems — EDA222/DIT161

Final exam, March 6, 2012 at 14:00 – 18:00 in the M building

**Examiner:**
Docent Jan Jonsson
Phone: 031–772 5220

**Aids permitted during the exam:**
J. Nordlander, *Programming with the TinyTimber kernel*
Chalmers-approved calculator

**Content:**
The written exam consists of 6 pages (including cover), containing 7 problems worth a total of 60 points.

**Grading policy:**
24–35 ⇒ grade 3
36–47 ⇒ grade 4
48–60 ⇒ grade 5

**Solution:**
Posted on the course home page on Wednesday, March 7, 2012 at 09:00.

**Results:**
Posted on the course home page on Tuesday, March 27, 2012 at 09:00.

**Inspection:**
Room 4128, Rännvägen 6 B, on Tuesday, March 27, 2012 at 13:00–15:00. Inspection at another occasion could be arranged by contacting the course examiner.

**Language:**
Your solutions should be written in English.

---

**IMPORTANT ISSUES**

1. Use separate sheets for each answered problem, and mark each sheet with the problem number.
2. Justify all answers. Lack of justification can lead to loss of credit even if the answer might be correct.
3. Explain all calculations thoroughly. If justification and method is correct then simple calculation mistakes do not necessarily lead to loss of credit.
4. If some assumptions in a problem are missing or you consider that the made assumptions are unclear, then please state explicitly which assumptions you make in order to find a solution.
5. Write clearly! If I cannot read your solution, I will assume that it is wrong.

---

Good Luck!

---

## PROBLEM 1

State whether the following propositions are True or False. Each correct statement will give 0.5 points; each erroneous statement will give -0.5 points; an omitted statement gives 0 points. Although a motivation for a correct answer is not required, a convincing one gives another 0.5 points, while an erroneous/weak one gives another -0.5 points. **Quality guarantee**: The total result for this problem cannot be less than 0 points. (6 points)

**a)** The RM-US priority-assignment policy has a utilization guarantee bound that converges towards 50% as the number of processors become very large.

**b)** An estimate of a task's worst-case execution time should be *tight* to avoid unnecessary waste of resources during scheduling of hard real-time tasks

**c)** Sporadic tasks can never be used in a real-time system with hard timing constraints.

**d)** The term *critical instant* in schedulability analysis refers to the latest point in time by which a task must have started its execution in order to meet its deadline.

**e)** Deadlock can never be completely avoided in systems where tasks require access to more than one exclusive resource at a time.

**f)** By *deadline inversion*, we mean a situation where the priority of a task is inversely proportional to the length of the relative deadline of the task.

---

## PROBLEM 2

The following questions are related to network communication.

**a)** Describe the underlying cause for message queuing delay in each of the following networks: TTP/C, FDDI, Ethernet and CAN. (2 points)

**b)** Describe how message transmission delay is a function of the properties of the message being sent and the network medium used. (2 points)

**c)** Describe why the Ethernet protocol is not suitable for use with hard real-time systems. (1 point)

**d)** Describe the *binary countdown* algorithm as used in CAN. (3 points)

---

## PROBLEM 3

Most scheduling analysis techniques assume the worst-case execution time (WCET) to model the computational demand of a real-time task. One of the earliest methods for WCET analysis was presented by Shaw in the end of the 1980s.

**a)** Assume that the function `main` (see below) is used as part of a real-time program and that the function, when called, is allowed to take at most 90 $\mu$s to execute. Derive WCET for function `main` by using Shaw's method and check whether the function's deadline will be met or not. Assume that each *declaration* and *assignment* statement costs 1 $\mu$s to execute. A *function call* costs 2 $\mu$s plus WCET for the function in question. Each *compare* statement costs 2 $\mu$s. Each *addition* and *subtraction* operation costs 3 $\mu$s. Each *multiplication* operation costs 5 $\mu$s. Each *return* statement costs 2 $\mu$s. All other language constructs can be assumed to take 0 $\mu$s to execute. (8 points)

```
int multiply(int a, int b) { return a * b; }

int methA(int a, int b) {
    int p;
    int i;

    p=a;
    i=1;

    if(b==0)
        return 1;

    while(i<b) {
        p=multiply(p,a);
        i=i+1;
    }

    return p;
}


int methB(int a, int b) {
    if(b==1)
        return a;
    else
        return multiply(a , methB(a, b-1));
}

int main() {
    char ans;
    int x;
    int y;

    x=2;
    y=3;

    if(methA(x,y) > methB(x,y)){
        ans='T';
        x=x+y;
    }
    else
        ans='F';

    return 1;
}
```

**b)** Identify two different false paths in the program given in subproblem a). (2 points)

---

## PROBLEM 4

With the TinyTimber kernel it is possible to implement periodic activities in a C program. Consider a real-time system with two independent periodic tasks. The table below shows $O_i$ (offset), $D_i$ (deadline) and $T_i$ (period) for the two tasks. All values are given in milliseconds.

| | $O_i$ | $D_i$ | $T_i$ |
|---|---|---|---|
| $\tau_1$ | 20 | 30 | 85 |
| $\tau_2$ | 0 | 55 | 105 |

On a separate sheet at the end of this exam paper you find a C-code template. Write your solutions to the following subproblems directly on that sheet and hand it in for grading together with the rest of your solutions. The code for the functions `Action1()` and `Action2()` is assumed to already exist.

**a)** Add code that makes the two methods `T1()` and `T2()` have the same timing behavior as the two periodic tasks, $\tau_1$ and $\tau_2$, mentioned above. (3 points)

**b)** Add code that measures the worst-case execution time of the function `Action1()` by taking time samples before and after the call to `Action1()` for each iteration of method `T1()` and keeping only the sample difference with the largest value. (2 points)

**c)** How are priorities (for run-time scheduling) assigned to the two periodic activities? (1 point)

---

## PROBLEM 5

Consider four tasks that share three exclusive resources $R_a, R_b$, and $R_c$. The tasks use the resources in the following way: Task $\tau_1$ first uses resource $R_a$ and then resource $R_c$. Task $\tau_2$ only uses resource $R_a$. Task $\tau_3$ only uses resource $R_b$. Task $\tau_4$ first uses resources $R_b$ and then resource $R_c$. The table below shows $H_{i,j}$, the maximum time (in milliseconds) that task $\tau_i$ may lock resource $R_j$ during its execution.

| | $H_{i,a}$ | $H_{i,b}$ | $H_{i,c}$ |
|---|---|---|---|
| $\tau_1$ | 3 | - | 2 |
| $\tau_2$ | 2 | - | - |
| $\tau_3$ | - | 1 | - |
| $\tau_4$ | - | 2 | 3 |

In order to reduce the priority inversion problem a priority-adjustment protocol must be used at run-time, for example, the Priority Inheritance Protocol (PIP) or the Priority Ceiling Protocol (PCP).

**a)** Describe PIP in terms of how it adjusts priorities at run-time. (2 points)

**b)** Describe PCP in terms of how it adjusts priorities at run-time. Also list the advantages of PCP compared to PIP. (5 points)

**b)** Derive a blocking factor $B_i$ for each task $\tau_i$ assuming that PCP is being used. Assume that rate-monotonic (RM) scheduling is being used and that the periods of the tasks are $T_1 = 9$, $T_2 = 24$, $T_3 = 15$, and $T_4 = 13$, respectively. (5 points)

---

## PROBLEM 6

Consider a real-time system with three independent periodic tasks and a run-time system that employs static scheduling using a time table. The table below shows $C_i$ (WCET), $D_i$ (deadline) and $T_i$ (period) for the three tasks. All tasks arrive the first time at time 0.

|  | $C_i$ | $D_i$ | $T_i$ |
|---|---|---|---|
| $\tau_1$ | 2 | 3 | 4 |
| $\tau_2$ | 2 | 6 | 8 |
| $\tau_3$ | 3 | 12 | 16 |

**a)** Describe how the run-time system works for time-table based scheduling. (3 points)

**b)** Contruct a time table for the execution of the three tasks $\tau_1$, $\tau_2$ and $\tau_3$. Assume that the tasks are allowed to preeempt each other. Your solution should clearly indicate the start and stop times for each task (or task segment, if a task is preempted). In addition, the total length of your time table (in time units) should be given. (6 points)

**c)** Evaluate the schedule described by your time table in sub-problem b). That is, does it constitute the best possible schedule, or does there exist a superior one? (1 point)

---

## PROBLEM 7

Consider a real-time system with three independent periodic tasks and a run-time system that uses preemptive global scheduling on $m = 2$ processors. The task priorities are given according to the deadline-monotonic (DM) policy. The table below shows $O_i$ (offset), $C_i$ (worst-case execution time), $D_i$ (deadline) and $T_i$ (period) for the three tasks.

|  | $O_i$ | $C_i$ | $D_i$ | $T_i$ |
|---|---|---|---|---|
| $\tau_1$ | 2 | 2 | 3 | 4 |
| $\tau_2$ | 4 | 4 | 5 | 6 |
| $\tau_3$ | 0 | 4 | 7 | 10 |

Use a suitable analysis method to determine whether the deadlines are met or not for the three tasks in the system. (8 points)

---

*Template code for Problem 4.*

---

```
#include TinyTimber.h

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

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




void T1(PeriodicTask *self, int u) {




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






}

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


}

void kickoff(PeriodicTask *self, int u) {



}

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