Real-Time Systems — EDA222/DIT161

Final exam, March 14, 2016 at 08:30 – 12:30 in the M building

Examiner:
Professor Jan Jonsson, Department of Computer Science and Engineering

Responsible teacher:
Jan Jonsson, phone: 031–772 5220
Visits the exam at 09:30 and 11:30

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 $\Rightarrow$ grade 3
36–47 $\Rightarrow$ grade 4
48–60 $\Rightarrow$ grade 5

Results:
When the grading is completed overall result statistics, and a time and location for inspection, will be announced on the course home page. Individual results will be posted in PingPong under 'Objectives & Progress'.

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 TinyTimber kernel supports concurrent execution of multiple methods, where at most one method at a time can be executed for each object.

b) In worst-case execution-time analysis, the behavior of an instruction cache is much more difficult to predict that the behavior of a data cache.

c) The response-time test for global fixed-priority multiprocessor scheduling is an exact feasibility test.

d) 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.

e) The immediate ceiling priority protocol (ICPP) is a deadlock free protocol.

f) If a given task set is known to be not schedulable, a *sufficient* feasibility test will always report the answer "no" when applied to that task set.

PROBLEM 2

The following questions are related to the CAN (Controller Area Network) technique.

a) State whether communication medium access in CAN is token-based or contention-based. (1 points)

b) Describe the message format used in CAN. (2 points)

c) 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. 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 195 μs to execute where.

- Each declaration and assignment statement costs 1 μs to execute.
- A function call costs 2 μs plus WCET for the function in question.
- Each compare statement costs 2 μs.
- Each addition and subtraction operation costs 3 μs.
- Each multiplication and division operation costs 4 μs.
- Each return statement costs 2 μs.
- Each modulo operation costs 5 μs.
- The function `abs()` is predefined and costs 5 μs. This function computes and returns the absolute value of the number given as its parameter. For example, `abs(-5)` returns 5. Assume that the overhead to call function `abs()` is already included in its WCET estimation.
- All other language constructs can be assumed to take 0 μs to execute.

```
int FuncA(int x, int y) {
    int z;
    z = 0;
    if (y == 0)
        z = 1;
    else {
        if (y == 1)
            z = x;
        else
            z = x * FuncA(x, y-1);
    }
    return z;
}

int FuncB(int x) {
    int y;
    y = x;
    if ((y % 3) != 0) {
        if (((y % 3) % 2) != 0)
            return y - 1;
        else
            return y + 1;
    }
    return y;
}

int main() {
    int a, b, c, d, e;
    int result, threshold;
    threshold = 100;
    result = -1;
    a = 3;
    b = 4;
    c = FuncA(abs(b), abs(a));
    if (c > threshold)
        e = c;
    else {
        d = FuncA(abs(a), abs(b));
        if (d > threshold)
            e = d;
        else
            e = c+d;
    }
    if (e > threshold)
        result = FuncB(e);
}
```

a) Derive WCET for function `main` by using Shaw's method and check whether the function's deadline will be met or not. (6 points)

b) What should be the maximum cost for `addition/subtraction` operations so that if the function's deadline is 195 μs, it can meet its dedline? (2 points)

c) Explain why is it preferred that WCET estimates for tasks in a real-time system are *pessimistic* as well as *tight*. (2 points)

PROBLEM 4

The TinyTimber kernel makes it possible to implement periodic activities in a C program. Consider a real-time system with four independent periodic tasks: three hard-real-time tasks (`T1`, `T2`, `T3`), and one soft-real-time background task `BG`.

The three hard-real-time tasks all arrive at $t = 0$ and have a common period of 2400 μs, but their execution is precedence constrained in the following way:

- First, task `T1` should execute for 300 μs. The relative deadline for task `T1` is 1600μs.
- Then, task `T2` should execute for 800 μs. The relative deadline for task `T2` is 1200μs.
- Finally, task `T3` should execute for 500 μs. The relative deadline for task `T3` is 2100μs.

The soft-real-time background task `BG` arrives at time $t = 0$, has a period of 1800 μs, and at each invocation executes for 700 μs. The deadline for the background task is equal to its period.

a) Construct a TinyTimber program with four methods `T1()`, `T2()`, `T3()`, and `BG()` that have the same timing behavior as the corresponding tasks mentioned above. On a separate sheet at the end of this exam paper you find a C-code template. Add the missing program code directly on that sheet and hand it in for grading together with the rest of your solutions. The code for the functions `Action300()`, `Action800()`, `Action500()`, and `Load700()` is assumed to already exist. (5 points)

b) Assuming that the TinyTimber kernel will be used to schedule the four tasks, is it possible to guarantee that tasks `T1`, `T2`, `T3` will meet their deadlines? Motivate your answer. Any timing overhead relating to function calls or the TinyTimber runtime system can be ignored. (3 points)

PROBLEM 5

Consider a real-time system with three independent periodic tasks and a run-time system that uses preemptive rate-monotonic (RM) scheduling. The table below shows O_i (offset), T_i (period) and U_i (utilization) for the three tasks. The value of the parameter T is not known.

	O_i	T_i	U_i
τ_1	0.6T	0.8T	1/4
τ_2	0	T	9/20
τ_3	0	1.5T	2/15

Use a suitable analysis method to determine the schedulability of the tasks in the system. (8 points)

PROBLEM 6

Consider a system with three periodic tasks and a run-time system employing preemptive deadline-monotonic (DM) scheduling. The table below shows C_i (WCET), D_i (deadline) and T_i (period) for the three tasks. The initial arrival time of each task is not known. All values are given in milliseconds.

	C_i	D_i	T_i
τ_1	6	10	31
τ_2	7	17	29
τ_3	10	25	47

The three tasks are not independent, but share three exclusive resources R_a, R_b, and R_c. The run-time system employs the Immediate Ceiling Priority Protocol (ICPP) to resolve resource request conflicts. The tasks use the resources in the following way:

- Task τ_1 first requests R_c and then, while using R_c, requests R_a
- Task τ_2 first requests R_c and then, while using R_c, requests R_b; then, after releasing the two resources, τ_2 requests R_a
- Task τ_3 first requests R_b and then, while using R_b, requests R_a; then, after releasing the two resources, τ_3 requests R_c

The table below shows $H_{i,j}$, the maximum time (in milliseconds) that task τ_i locks resource R_j during its execution.

	$H_{i,a}$	$H_{i,b}$	$H_{i,c}$
τ_1	2	-	2
τ_2	1	2	2
τ_3	2	3	2

Use a suitable analysis method to determine the schedulability of the tasks in the system. Note that nested blocking is used by all tasks. This could lead to accumulated critical region blocking times in the final blocking factor. (10 points)

PROBLEM 7

There are two approaches for scheduling tasks on multiprocessor platform: the *partitioned* approach and the *global* approach.

a) State one advantage and one disadvantage of global scheduling algorithm. (2 points)

b) Consider a task set with total utilization equal to 1.25. Can you guarantee that all the deadlines of the task set will be met using RM-US scheduling algorithm on $m = 3$ processors? Why or why not? (3 points)

c) The table below shows C_i (WCET) and T_i (period) for four periodic tasks to be scheduled on $m = 2$ processors using rate-monotonic first-fit (RMFF) partitioned scheduling algorithm. The relative deadline of each periodic task is equal to its period.

	C_i	T_i
τ_1	60	100
τ_2	?	20
τ_3	$2C_2$	40
τ_4	5	10

i) Consider that the WCET of task τ_3 is twice the WCET of task τ_2, i.e., $C_3 = 2C_2$. Determine the largest integer value of C_3 by assigning the tasks on $m = 2$ processors such that all the deadlines are met. Show the assignment of the tasks to the processors. (5 points)

ii) Consider a new task τ_5 where $C_5 = 5$ and $T_5 = 100$. Given the tasks-to-processors assignment you obtained in part i), can you assign task τ_5 to some processor so that all the five tasks meet their deadlines? Why or why not? (2 points)

Excerpt of program code for Problem 4 a).

```
#include "TinyTimber.h"

typedef struct {
    Object super;
} TaskObj;

TaskObj A = { initObject() };
TaskObj B = { initObject() };

void T1(TaskObj, int);
void T2(TaskObj, int);
void T3(TaskObj, int);

void T1(TaskObj *self, int u) {


    Action300(); // Do work for 300 microseconds


}

void T2(TaskObj *self, int u) {


    Action800(); // Do work for 800 microseconds


}

void T3(TaskObj *self, int u) {


    Action500(); // Do work for 500 microseconds


}

void BG(TaskObj *self, int u) {


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


}

void kickoff(TaskObj *self, int u) {



}

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