Real-Time Systems — EDA223/DIT162

Final exam, March 18, 2019 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

Electronic dictionaries may not be used.

Content:
The written exam consists of 6 pages (including cover and hand-in sheet), containing 7 problems worth a total of 60 points.

Grading policy:

24–35 points ⇒ grade 3	24–43 points ⇒ grade G (GU)
36–47 points ⇒ grade 4	
48–60 points ⇒ grade 5	44–60 points ⇒ grade VG (GU)

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 available in Ladok.

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 we cannot read your solution, we will assume that it is wrong.
6. A hand-in sheet is available at the end of the exam script. Do not forget to submit it together with your other solution sheets!

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) For a *sporadic* task there is a maximum time interval between two, subsequent, arrivals that is guaranteed to never be exceeded.

b) TinyTimber's AFTER() construct enables scheduling of periodic tasks without systematic time skew.

c) For an NP-complete problem to have *pseudo-polynomial* time complexity the largest number in the problem must be bounded by the input length (size) of the problem.

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

e) The term *false path* in execution-time analysis refers to a program subroutine that always returns the Boolean value 'False' when running the program.

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

In real-time kernels, such as TinyTimber, access to sensitive data structures and I/O devices is protected by means of *machine-level mutual exclusion* where special support from the processor is required.

a) State what type of machine-level support is required for guaranteeing mutual exclusion in *single-processor systems*, and describe how the approach can guarantee mutual exclusion. Also, explain why this approach is not typically used for multi-processor systems. (4 points)

b) State what type of machine-level support is required for guaranteeing mutual exclusion in *multi-processor systems*, and describe how the approach can guarantee mutual exclusion. (4 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 `Control` (see below) is used as part of a real-time program and that the function, when called, is allowed to take at most 73 μs to execute.

The system has one input port, located at address 0x40021010, and one output port, located at address 0x40020C14. The input port is connected to a sensor that delivers 8-bit values in the range $[-9, +9]$.

```
#define Inport  (*((volatile signed char *)(0x40021010)))
#define Outport (*((volatile signed char *)(0x40020C14)))
```

```
int Calc(int x) {
   int i;
   int r;
   i = 0;
   r = x;
   while (i < 3) {
      r = r * x;
      i = i + 1;
   }
   r = r - 1;
   return r;
}

void Control() {
   int c;
   int r;
   c = Inport;
   r = Calc(c) / 3;
   if (r <= 800)
       r = r >> 3;
   else
       r = (3 * r) / 289 + 2;
   Outport = r;
}
```

Use Shaw's method to solve sub-problems a), b) and c). To that end, assume the following costs:

- Each declaration and assignment statement costs 1 μs to execute.
- A function call costs 2 μs plus WCET for the function in question.
- Each evaluation of the logical condition in an `if`-statement or a `while`-statement costs 2 μs.
- Each 3-bit shift operation ('>> 3') costs 2 μs.
- Each add and subtract operation costs 3 μs.
- Each multiply operation costs 5 μs.
- Each divide operation costs 8 μs.
- Each return statement costs 2 μs.
- All other language constructs can be assumed to take 0 μs to execute.

a) Show that function `Control` will not meet its deadline of 73 μs with the code given above and an input port value range of $[-9, +9]$. (6 points)

b) What is the largest input port data range for which function `Control` will be able to meet its deadline of 73 μs with the code given above? (3 points)

c) A colleague of yours suggests that you replace the code for the subroutine `Calc` with the code given below, which is functionally equivalent to the old code. Is the new code a good replacement from a timing point of view? (3 points)

```
int Calc(int x) {
   int r;
   r = x * x - 1;
   r = r * (r + 2);
   return r;
}
```

PROBLEM 4

The TinyTimber kernel makes it possible to implement periodic activities in a C program. Consider a real-time system with three periodic tasks: `T1`, `T2`, and `T3`.

The three tasks all arrive at time $t = 0$ and have a common period of 2300 μs, but their execution is precedence constrained in the following way:

- First, task `T1` should execute for 400 μs. The relative deadline for task `T1` is 700μs.
- Then, task `T2` should execute for 600 μs. The relative deadline for task `T2` is 1500μs.
- Finally, task `T3` should execute for 700 μs. The relative deadline for task `T3` is 2000μs.

You should write a TinyTimber program implementing the execution of three methods `T1()`, `T2()`, and `T3()` corresponding to the tasks mentioned above. On a separate sheet at the end of this exam paper you find a C-code template. For sub-problems a) and b) below 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 `Action400()`, `Action600()`, and `Action700()` is assumed to already exist.

a) Implement the code for the execution of the three methods `T1()`, `T2()`, and `T3()` so that they have the same timing behavior as the corresponding tasks mentioned above. Add the missing TinyTimber statements directly in the provided C-code template. (3 points)

b) Extend the program with code that measures the worst-case execution time of the method chain `T1()`, `T2()`, and `T3()`. For each iteration of the method chain the sum of actual execution times for the functions `Action400()`, `Action600()`, and `Action700()` should be calculated, and only the largest value of this sum should be stored. Timing overhead relating to the TinyTimber runtime system should not be included in the measurements of the worst-case execution time. Add the missing TinyTimber statements directly in the provided C-code template. (5 points)

PROBLEM 5

Consider a real-time system with n independent periodic tasks and a run-time system that employs pre-emptive single-processor scheduling using the deadline-monotonic (DM) priority-assignment approach. The C_i (WCET), D_i (relative deadline) and T_i (period) of each task τ_i are such that $C_i \leq D_i \leq T_i$. All tasks arrive at time $t = 0$. Assume for sub-problems a), b) and c) below that $\forall j \in hp(i) : D_j < D_i$.

a) Consider the standard response-time analysis given in Eq. (1):

$$R_i^{k+1} = C_i + \sum_{\forall j \in hp(i)} \left\lceil \frac{R_i^k}{T_j} \right\rceil C_j \qquad \forall i : R_i \leq D_i \tag{1}$$

Can we guarantee that a higher-priority task τ_j always meets its deadline according to the analysis in Eq. (1) if a lower-priority task τ_i meets its deadline according to the analysis in Eq. (1)? Motivate your answer. (2 points)

b) Now, consider a new response-time analysis given in Eq. (2):

$$R_i^{k+1} = C_i + \sum_{\forall j \in hp(i)} \left\lceil \frac{R_i^k}{(T_j - \alpha)} \right\rceil C_j \qquad \forall i : R_i \leq D_i \tag{2}$$

Assume that α is a non-negative integer, and it applies for each task τ_j that $T_j > \alpha$. Determine to what extent the new response-time analysis in Eq. (2) can be used to determine schedulability of the tasks. (3 points)

c) The table below shows C_i (WCET), D_i (relative deadline) and T_i (period) for a task set with $n = 3$ periodic tasks.

	C_i	D_i	T_i
τ_1	4	10	10
τ_2	5	18	18
τ_3	4	20	20

Apply the new response-time analysis in Eq. (2) to determine the response time of each task, assuming that $\alpha = 2$. What can you conclude about the schedulability of the tasks? (5 points)

PROBLEM 6

Consider a real-time system with three independent periodic tasks and a run-time system that employs preemptive single-processor scheduling using the earliest-deadline-first (EDF) priority-assignment approach. The table below shows C_i (WCET), D_i (relative deadline) and T_i (period) for the three tasks. All tasks arrive at time $t = 0$.

	C_i	D_i	T_i
τ_1	C_1	8	10
τ_2	2	6	20
τ_3	$2C_1$	10	40

Apply processor-demand analysis to determine the largest positive integer value of C_1 for which all the tasks meet their deadlines. (8 points)

PROBLEM 7

Consider a real-time system with n independent periodic tasks that should be scheduled on a multiprocessor system, using preemptive static-priority scheduling. To that end, there are two possible approaches: global scheduling and partitioned scheduling.

a) Describe the underlying causes why so few results from single-processor scheduling theory can be used for global scheduling on a multiprocessor system. (4 points)

b) The table below shows C_i (WCET) and T_i (period) for a task set with $n = 5$ periodic tasks. The relative deadline of each periodic task is equal to its period. All tasks arrive at time $t = 0$.

	C_i	T_i
τ_1	4	20
τ_2	5	5
τ_3	12	40
τ_4	3	10
τ_5	20	100

As a system designer you may choose to use either global scheduling or partitioned scheduling to execute the five tasks on a multiprocessor system. If your design objective is to use as few processors as possible, while still guaranteeing that all deadlines of the tasks are met, which approach (i.e., global or partitioned scheduling) would you choose? Motivate your answer. (4 points)

```
#include "TinyTimber.h"

typedef struct {
    Object super;
} TaskObj;

TaskObj A = { initObject() };


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

void T1(TaskObj *self, int u) {



    Action400(); // Do work for 400 microseconds



}

void T2(TaskObj *self, int u) {



    Action600(); // Do work for 600 microseconds



}

void T3(TaskObj *self, int u) {



    Action700(); // Do work for 700 microseconds



}

void kickoff(TaskObj *self, int u) {



}

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