Real-Time Systems — EDA223/DIT162

Final exam, March 16, 2020 at 08:30 – 12:30 in your home

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

Consultation:
Questions during the exam should be sent to the examiner via email or Canvas messaging. Formulate your question in as precise way as possible to facilitate a correspondingly precise answer. Include your phone number in the message, so the examiner can call you back should the answer require an interactive discussion.

Aids permitted during the exam:
No restrictions.

Content:
The written exam consists of 7 pages (including cover and program code 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. Submit a separate solution file for each answered problem, and mark each page in the file with your name and personnummer, and 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. Imported images of handwritten solutions must be of high quality. If we cannot read your solution, we will assume that it is wrong.

Good Luck!

PROBLEM 1

The following questions are related to uniprocessor scheduling using the rate-monotonic (RM) priority-assignment policy.

a) Liu & Layland's utilization-based test for the RM policy was originally intended for strictly periodic tasks. Explain why the test is also applicable for sporadic tasks. (2 points)

b) Consider a real-time system with two periodic tasks, where task priorities are assigned according to the RM policy. The parameters C_i (WCET) and T_i (period) describe the timing properties of each task τ_i. The relative deadline of each task is equal to its period. Both tasks arrive at time $t = 0$.

For task τ_1 it is known that $C_1/T_1 = 0.4$.

For task τ_2 it it is known that $C_2 = 15\,C_1$ and $T_2 = 10\,T_1$.

Use a suitable analysis method to determine whether the deadlines are met or not for the two tasks. (4 points)

PROBLEM 2

The following questions are related to optimality and time complexity of scheduling algorithms.

a) The rate-monotonic (RM) and earliest-deadline-first (EDF) priority-assignment policies are both optimal under similar assumptions regarding the task model on a uniprocessor system. Does this mean that the policies are equally good in terms of schedulability, or does one policy dominate the other? (2 points)

b) Assume that a synchronous task set Γ is known to be schedulable on a uniprocessor using the deadline-monotonic (DM) priority-assignment policy. If one of the tasks in Γ would increase its offset by one time unit, keeping its other task parameters intact, would task set Γ still be schedulable? Motivate your answer! (2 points)

c) The response-time analysis is known to be an NP-complete problem with pseudo-polynomial time complexity. Refer to the conditions that must apply for such a problem to have pseudo-polynomial time complexity, and show that response-time analysis fulfils these conditions, when assuming periodic task sets where all tasks arrive at time $t = 0$ and the deadline of a task does not exceed its period. (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 `Brake` (see below) is used as part of a real-time program and that the function, when called, is allowed to take at most 87 μs to execute.

The system has two 8-bit input ports, located at addresses 0x40051410 and 0x40051810, and one 8-bit output port, located at address 0x40014C10. The input ports are connected to two different sensors that both deliver values in the range $[1, 15]$.

```
#define Inport_Velocity  (*((volatile unsigned char *)(0x40051410)))
#define Inport_Pedal     (*((volatile unsigned char *)(0x40051810)))
#define Outport_Force    (*((volatile unsigned char *)(0x40014C10)))
```

```
int Calculate(int x, int y) {
   int result;
   result = 0;
   while (x != 0) {
      if ((x & 1) == 1)
          result = result + y;
      x = x >> 1;
      y = y << 1;
   }
   return result;
}

void Brake() {
   int v;
   int p;
   int f;
   v = Inport_Velocity;
   p = Inport_Pedal;
   f = Calculate(v, p);
   Outport_Force = f;
}
```

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.
- Each function call costs 7 μs plus WCET for the function in question.
- Each evaluation of a comparison ('`!=`' or '`==`') costs 2 μs.
- Each bitwise–shift operation ('`>> 1`' or '`<< 1`') costs 3 μs.
- Each bitwise–and operation ('`&`') costs 1 μs.
- Each multiply operation costs 5 μs.
- Each add operation costs 3 μs.
- Each return statement costs 2 μs.
- All other language constructs can be assumed to take 0 μs to execute.

a) Show that function `Brake` will not meet its deadline of 87 μs with the code given above and the input port value ranges of $[1, 15]$. (6 points)

b) Derive the largest input port range for `Input_Velocity` for which function `Brake` will be able to meet its deadline of 87 μs with the code given above? (3 points)

c) Replace the code for the subroutine `Calculate` with the code given below, which is functionally equivalent. Is the new code a good replacement from a timing point of view? (3 points)

```
int Calculate(int x, int y) {
   int result;
   result = 0;
   while (x != 0) {
      result = result + y * (x & 1);
      x = x >> 1;
      y = y << 1;
   }
   return result;
}
```

PROBLEM 4

The TinyTimber kernel makes it possible to implement periodic activities in a C program. Consider a real-time system with two independent periodic tasks, `T1` and `T2`, with hard real-time constraints.

The timing properties of the tasks are specified as follows:

- The tasks are synchronous, and arrive for the first time at $t = 0$.
- Task `T1` should be invoked every 9 ms, and at each invocation execute for 4 ms (by calling a function `Action4ms()`). The relative deadline for task `T1` is 8 ms.
- Task `T2` should be invoked every 4 ms, and at each invocation execute for 2 ms (by calling a function `Action2ms()`). The relative deadline for task `T2` is 4 ms.

a) Construct a timing diagram for the execution of the tasks, assuming that the task are scheduled on one processor by the TinyTimber kernel. The timing diagram should be based on the desired behavior of the task set (as given by the specification above) and should span the interval $[0, P]$, where P is the hyper period of the task set. (2 points)

b) On a separate sheet at the end of this exam paper you find a C-code program, that makes an (unsuccessful) attempt to implement the desired behavior of the task set. The code for the C functions `Action2ms()` and `Action4ms()` is assumed to already exist, and to be working correctly.

Identify the design errors (there are several) in the given program that prevents it from producing the desired behavior, and describe briefly how they produce the error, and how to correct them. By 'design errors' we mean flaws not on the syntax level of the code, but more fundamental issues that, even if the program compiled without any errors, would cause an erroneous behavior. Copy the code from the supplied sheet to your solution file, and modify the code accordingly to show how it should have been written. (4 points)

PROBLEM 5

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

	C_i	D_i	T_i
τ_1	7	19	19
τ_2	6	17	28
τ_3	6	25	30

The three tasks are not independent, but share two exclusive resources R_a and R_b. Assume that the run-time system employs the Immediate Ceiling Priority Protocol (ICPP) to resolve resource request conflicts. Each task uses the resources in the following way during each of its periodic invocations:

- Task τ_1 first requests R_a; and after releasing R_a, requests R_b; and then releases R_b.
- Task τ_2 first requests R_b; and after releasing R_b, requests R_a; and then releases R_a.
- Task τ_3 requests R_a, and then releases R_a.

The table below shows $H_{i,j}$, the maximum time that task τ_i may lock resource R_j during its execution. Note: each resource use $H_{i,j}$ is assumed to be included in the normal execution time, C_i.

	$H_{i,a}$	$H_{i,b}$
τ_1	5	1
τ_2	1	4
τ_3	3	-

a) Determine, using a suitable analysis method, the schedulability of the task set. (4 points)

b) What is the smallest integer value of the deadline, D_3, of task τ_3 for which the task set is schedulable? Motivate your solution. (4 points)

PROBLEM 6

Consider a real-time system with three 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 (deadline) and T_i (period) for each task τ_i. All tasks arrive at $t = 0$.

	C_i	D_i	T_i
τ_1	3	5	8
τ_2	3	13	16
τ_3	11	19	32

a) Perform processor-demand analysis to determine the schedulability of the task set. (5 points)

b) Verify the outcome of the analysis in sub-problem a) by constructing a timing diagram for the execution of the three tasks that spans the hyper period of the task set. (2 points)

c) What is the largest integer value of the WCET, C_2, of task τ_2 for which the task set is schedulable? Motivate your solution. (3 points)

PROBLEM 7

The following sub-problems are related to multiprocessor scheduling of independent periodic tasks.

a) Consider a task set with four independent periodic tasks and a run-time system that uses preemptive *global scheduling* on $m = 3$ processors. The task priorities are given according to the rate-monotonic (RM) policy. The table below shows C_i (WCET) and T_i (period) for each task in the task set. The relative deadline of each task is equal to its period. All tasks arrive at time $t = 0$.

	C_i	T_i
τ_1	10	100
τ_2	10	100
τ_3	10	100
τ_4	181	200

Show that is it is not possible to schedule the given task set on $m = 3$ processors such that all task deadlines are met, when using global scheduling with RM priorities. (3 points)

b) Show, by using a suitable *processor utilization test for global scheduling*, that there exists a better approach to give static priorities to the tasks in sub-problem a) such that all task deadlines will be met when using global scheduling on $m = 3$ processors. (2 points)

c) Consider a task set with six independent periodic tasks and a run-time system that uses preemptive *partitioned scheduling* on $m = 3$ processors. The task priorities and task-to-processor assignments are given according to the the rate-monotonic first-fit (RMFF) algorithm. The table below shows C_i (WCET) and T_i (period) for each task in the task set. The relative deadline of each task is equal to its period. All tasks arrive at time $t = 0$.

	C_i	T_i
τ_1	10	100
τ_2	10	100
τ_3	10	100
τ_4	141	200
τ_5	141	200
τ_6	141	200

Show that is it is not possible to schedule the given task set on $m = 3$ processors such that all task deadlines are met, when using the RMFF partitioned scheduling algorithm. (3 points)

d) Show that there exists a better task-to-processor assignment for the tasks in sub-problem b) such that all task deadlines will be met when using partitioned scheduling on $m = 3$ processors and giving task priorities according to the rate-monotonic (RM) policy. (2 points)

Sheet with program code for Problem 4.

```
#include "TinyTimber.h"

Object ptask = initObject();



void T1(Object *self, int u) {

    Action4ms();  // C function doing time-critical work for 4 ms

    AFTER(MSEC(8), self, T1, 0);

}

void T2(Object *self, int u) {

    Action2ms();  // C function doing time-critical work for 2 ms

    AFTER(MSEC(4), self, T2, 0);

}

void kickoff(Object *self, int u) {

    ASYNC(self, T1, 0);

    ASYNC(self, T2, 0);

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