Introduction to Real-Time Systems — LET627

Final exam, May 26, 2018 at 14:00 – 18:00 in the Saga building

---

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

Responsible teacher:
Jan Jonsson, phone: 031–772 5220
Visits the exam at 15:00, and then at several occasions.

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

Content:
The written exam consists of 8 pages (including cover, list of equations and hand-in sheet), containing 7 problems worth a total of 60 points.

Grading policy:
24–35 points ⇒ grade 3
36–47 points ⇒ grade 4
48–60 points ⇒ 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 can be written in Swedish or 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) By *deadline inversion*, we mean a situation where the priority assigned to a task is inversely proportional to the deadline of the task.

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

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 RM-US priority-assignment policy has a utilization guarantee bound that converges towards 41% as the number of processors become very large.

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

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 systems that employ concurrent execution of multiple tasks with shared resources there is a potential risk that *deadlock* may occur.

a) State the four conditions for deadlock to occur in such systems. (4 points)

b) In such systems where tasks are assigned static priorities it is possible to avoid deadlock by using a run-time protocol that supports *ceiling priorities*. Describe the basic idea of a priority ceiling protocol (such as ICPP). (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 `Respond` (see below) is used as part of a real-time program and that the function, when called, is allowed to take at most 80 μs to execute.

The system has one 8-bit input port, located at address 0x40021010, and one 16-bit output port, located at address 0x40020C10. The input port is connected to a sensor that delivers values in the range $[0, +6]$.

```
#define Inport  (*((volatile signed char *)(0x40021010)))
#define Outport (*((volatile signed char *)(0x40020C10)))
```

```
int Calculate(int x) {
   int t;

   if (x == 0) {
      t = 1;
   } else {
      if (x == 1) {
         t = 1;
      } else {
         t = x * Calculate(x-1);
      }
   }

   return t;
}

void Respond() {
   int c;
   int r;

   c = Inport;

   r = (Calculate(c) << 4) + c;

   Outport = r;
}
```

Use Shaw's method to solve sub-problems a) and b). 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 costs 2 μs.
- Each 4-bit shift operation ('$<<$ 4') costs 2 μs.
- Each add and subtract operation costs 3 μs.
- Each multiply operation costs 5 μs.
- Each return statement costs 2 μs.
- All other language constructs can be assumed to take 0 μs to execute.

a) Show that function `Respond` is not able to meet its deadline of 80 μs for the given input port value range of $[0, +6]$. (7 points)

b) Derive the largest input port data range for which function `Respond` will be able to meet its deadline of 80 μs. (3 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: `T1` and `T2`, with the following properties:

- Task `T1` has a period of 105 ms, and should start executing for the first time at time t = 0 ms at the earliest. The relative deadline for task `T1` is 30 ms.
- Task `T2` has a period of 70 ms, and should start executing for the first time at time t = 15 ms at the earliest. The relative deadline for task `T2` is 25 ms.

You should write a TinyTimber program implementing the execution of two methods `T1()` and `T2()` 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-problem a) 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 `Action1()` and `Action2()` is assumed to already exist.

a) Implement the code for the execution of the two methods `T1()` and `T2()` 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)

Some general questions relating to the TinyTimber run-time system:

b) Describe how priorities are assigned to scheduled activities (such as the two periodic tasks above) in TinyTimber. (2 points)

c) State what type of shared-resource protocol is used by the TinyTimber run-time system. (1 point)

PROBLEM 5

Consider a real-time system with three independent periodic tasks and a run-time system that uses preemptive single-processor using the rate-monotonic (RM) priority-assignment approach. The table below shows O_i (offset), C_i (WCET) and U_i (utilization) for the three tasks. The relative deadline of each periodic task is equal to its period. The value of the parameter C is not known.

	O_i	C_i	U_i
τ_1	0.4C	0.2C	1/3
τ_2	0	0.3C	3/8
τ_3	0	0.3C	1/4

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

PROBLEM 6

Consider a real-time system with three periodic tasks and a run-time system that employs preemptive single-processor 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 time $t = 0$.

	C_i	D_i	T_i
τ_1	3	4	8
τ_2	4	12	16
τ_3	8	30	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 from time $t = 0$ to time $t = LCM(T_1, T_2, T_3)$. (2 points)

c) 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. (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) 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)

c) Describe the common property of the given task sets in sub-problems a) and b) that causes both global scheduling and partitioned scheduling to fail to meet task deadlines? (2 points)

d) Show, by using a suitable *processor utilization* analysis, that there exists a better approach to give 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)

e) 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)

List of useful expressions and equations.

$$n\left(2^{1/n}-1\right)$$

$$m\left(2^{1/2}-1\right)$$

$$\frac{m^2}{3m-2}$$

$$R_i^{n+1} = C_i + \sum_{\forall j \in hp(i)} \left\lceil \frac{R_i^n}{T_j} \right\rceil C_j$$

$$C_P(0,L) = \sum_{i=1}^{n} \left(\left\lfloor \frac{L-D_i}{T_i} \right\rfloor + 1 \right) C_i$$

$$R_i^{n+1} = C_i + \frac{1}{m} \sum_{\forall j \in hp(i)} \left(\left\lceil \frac{R_i^n}{T_j} \right\rceil \cdot C_j + C_j \right)$$

```
#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);
}
```