Real-Time Systems — EDA222/DIT161

Final exam, March 16, 2015 at 08:30 – 12:30 in the V building

---

Examiner:

Professor 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

Solutions:

Posted on the course home page on Tuesday, March 17, 2015 at 09:00.

Results:

Posted on the course home page on Tuesday, March 31, 2015 at 09:00.

Inspection:

Room 4128, Rännvägen 6 B, on Tuesday, March 31, 2015 at 10:00–12:00. 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) TinyTimber's AFTER() construct enables scheduling of periodic tasks without systematic time skew.

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

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) 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 facilities for machine level mutual exclusion help implementing higher-level mutual exclusion techniques, such as monitors, semaphores, and protected objetcs.

a) State a machine-level technique to guarantee mutual exclusion in *single-processor systems*, and what is important to consider when this technique is applied (to prevent inefficiency). Also, explain why this technique is not typically used for multi-processor systems. (3 points)

b) In *multi-processor systems* with shared memory, a test-and-set instruction is used for handling critical regions. Describe how this operation guarantees mutual exclusion. (2 points)

c) Describe the functionality of the test-and-set operation in the form of a procedure Test_and_Set in the C or Ada language. The parameters needed for the procedure must be clearly stated. (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 222 μs to execute. Also assume that:

- 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){
    if(x == 1)
        return x;
    else
        return abs(x) * FuncA(abs(x) - 1);
}

int FuncB(int a, int b){
    if(b == 0)
        return a;
    else
        return FuncB(b , a % b);
}

int main(){
    char Flag;
    int result;
    int P;
    int Q;
    Q = -4;

    if((FuncA(Q) % 2) == 0)
        P = FuncA(Q) - 1;
    else
        P = FuncA(Q);
    if((P % abs(Q)) == 0)
        result = abs(Q);
    else
        result = FuncB(P , abs(Q));
    if (result != 1)
    {
        Flag = 'F';
        result = P;
    }
    else
        Flag = 'T';
    return result;
}
```

a) Derive WCET for function `main` by using Shaw's method and check whether the function's deadline will be met or not. (8 points)

b) Now assume that the deadline of function `main` is 212 μs. What would then be the largest cost for an `abs()` operation in order for the function to meet its deadline? (2 points)

PROBLEM 4

The TinyTimber kernel makes it possible to implement periodic activities in a C program. Consider a real-time system with three independent tasks: two tone generator tasks, and one background task.

Tone generator `T1` should generate a square wave of 1000 Hz by alternately flipping ('0'-to-'1' and '1'-to-'0', respectively) bit 0 of an output port (by calling method 'FlipBit()' in object 'IOPort' with parameter '0'). Tone generator `T2` should generate a square wave of 2500 Hz by alternately flipping bit 1 of the same output port (by calling method 'FlipBit()' in object 'IOPort' with parameter '1'). The deadline of both tone generator methods should be 50 μs.

The background task should be invoked every 1300 μs, and at each invocation execute for 800 μs (by calling a subroutine 'Action800()'). The deadline for the background task is equal to its period.

On a separate sheet at the end of this exam paper you find an excerpt of a C program that implements the periodic tasks. The code excerpt assumes that all necessary library files have been included for TinyTimber and IOPort functionality, as well as for subroutine 'Action800()'.

a) Identify two design errors in the given program that will prevent it from producing tones with the intended frequencies, and describe how to correct the errors. 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. (4 points)

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

b) How is mutual exclusion achieved in TinyTimber? (1 point)

c) 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 rate-monotonic (RM) scheduling. The table below shows O_i (offset), C_i (worst-case execution time), T_i (period), and D_i (deadline) for the three tasks.

	O_i	C_i	T_i	D_i
τ_1	0	4	30	30
τ_2	12	4	16	16
τ_3	0	9	20	20

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

PROBLEM 6

Consider a real-time system in a control application, with three independent periodic tasks. The table below shows C_i (WCET) and T_i (period) for the three tasks. For all task it applies that the deadline $D_i = T_i$. All tasks arrive the first time at time 0. The tasks are allowed to preeempt each other.

	C_i	T_i
τ_1	2	5
τ_2	4	13
τ_3	6	29

a) Assume a run-time system that employs dynamic scheduling using the rate-monotonic (RM) priority assignment policy. Show that the task set is schedulable using RM. (3 points)

b) Now, assume a run-time system that employs static scheduling using a time table. In order to generate a compact cyclic time table (that is, with as few task instances per cycle as possible) for the task set it would be preferred to be able to adjust one or more of the task periods. Luckily, the control properties of the application allows the periods of tasks τ_2 and τ_3 to deviate at most ± 3 time units from the original value given in the table above. The control properties of the application also dictate that the execution of task τ_1 must not experience any jitter. Therefore, the period of task τ_1 cannot be changed, and the task must always execute as soon as it arrives.

Choose a suitable version of the task set given above (possibly modifying the period for tasks τ_2 and/or τ_3 within the given bounds), and construct a compact cyclic time table for the execution of the task set version. Make sure that the chosen task set version is still schedulable if the original periods are being modified. Your solution should clearly indicate the start and stop times for each task instance (or task segment, if a task is preempted). In addition, the total length of your time table (in time units) should be given. (7 points)

PROBLEM 7

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

a) Describe two underlying causes to explain why few results from the uniprocessor case can be used for analyzing global fixed-priority scheduling. (4 points)

b) State two reasons why the response-time analysis for global fixed-priority scheduling is a sufficient schedulability test only, and not an exact test. (2 points)

c) The table below shows C_i (WCET) and T_i (period) for six periodic tasks to be scheduled on $m = 3$ processors using rate-monotonic first-fit (RMFF) partitioned scheduling algorithm. The relative deadline of each periodic task is equal to its period. By assigning the tasks on $m = 3$ processors according to RMFF, determine the smallest integer value of T_5 such that all the deadlines are met. Show the assignment of the tasks on the processors. (6 points)

	C_i	T_i
τ_1	160	200
τ_2	10	25
τ_3	10	40
τ_4	5	20
τ_5	10	?
τ_6	5	10

Excerpt of program code for Problem 4.

```
typedef struct {
    Object super;
} TaskObj;

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

void T1(TaskObj *self, int u) {
    SYNC(IOPort, FlipBit, 0); // Flip bit on bit 0 on IOPort
    SEND(USEC(500), USEC(50), self, T1, 0);
}

void T2(TaskObj *self, int u) {
    SYNC(IOPort, FlipBit, 1); // Flip bit on bit 1 on IOPort
    SEND(USEC(250), USEC(50), self, T2, 0);
}

void BG(TaskObj *self, int u) {
    Action800(); // Do background work for 800 microseconds
    SEND(USEC(1300), USEC(1300), self, BG, 0);
}

void kickoff(TaskObj *self, int u) {
    SEND(USEC(0), USEC(1300), &A, BG, 0); // Background task
    SEND(USEC(0), USEC(50), &A, T1, 0); // Tonegenerator for 1000 kHz
    SEND(USEC(0), USEC(50), &B, T2, 0); // Tonegenerator for 2500 kHz
}

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