Dependable Real-Time Systems — EDA423 / DIT173

Homework assignment #2, 2019/20 — Due no later than 23:59, May 29, 2020

Content:

The homework assignment consists of 8 pages (including cover), containing 8 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)

Late submission policy:

Solutions report is late by $[10 \ldots 60)$ minutes ⇒ 25% reduction of points
Solutions report is late by $[1 \ldots 4)$ hours ⇒ 50% reduction of points
Solutions report is late by ≥ 4 hours ⇒ 100% reduction of points

Language:

Your solutions should be written in English.

IMPORTANT ISSUES

1. As part of the solution for each assignment problem, every member of your group *must declare* her/his percentage of contribution in deriving the solution of that particular problem. Each member *must agree* for each assignment problem the percentage declared by the other member. A student's score for the solution of each assignment problem will be at most her/his declared percentage multiplied by the total points awarded for that problem.
2. The homework assignment should be solved in groups of no more than 2 (two) students per group.
3. Justify all answers. Lack of justification can lead to loss of credit even if the answer might be correct. Explain all calculations thoroughly.
4. If some assumptions in a problem are missing or you consider that the made assumptions are unclear, discuss this with the examiner on a consultation session. If the examiner informs you that the problem is deliberately formulated with some assumptions left open you should state explicitly in your solutions report which assumptions you make in order to find a solution to that problem.
5. The solutions report must be computer generated and contain a cover page which states your group number and, for each group member, the name and social security number ("personnummer").
6. The solutions report should be handed in as an electronic document, submitted via the course home page. The document must be in PDF format.
7. In conjunction with the submission of the solutions report, each member of your group should book a time with the course examiner for the oral examination of the solutions.
8. At the oral examination the student should summarize, and argue for, the submitted solutions. NOTE: Each student must be able to defend any of their group's submitted solutions.
9. The course examiner will assess, and award points for, the solutions and the argumentation by the student. Individual student grades will be awarded, not group grades!
10. It is STRICTLY FORBIDDEN to plagiarize other solutions. Each group should construct their own solutions.

Good Luck!

PROBLEM 1

In most schedulability theory preemption incurs no overhead. Let us now relax that assumption.

Consider a task set with two tasks where each task τ_i is characterized by (C_i, T_i), such that $C_i \leq T_i$ (using the notation from the lecture notes). The relative deadline of each task is equal to its period.

The tasks should be scheduled on a uniprocessor using preemptive rate-monotonic (RM) scheduling, assuming a work-conserving scheduler.

Assume that each preemption has an overhead of x, where x is expressed in time units. Given C_1, C_2, T_1, and T_2, obtain the maximum value of x for which the task set is RM-schedulable. Your analysis should originate from an *exact* feasibility test. If your obtained maximum value of x is not exact you should include a motivation for each approximation made in the analysis. Hint: make sure to validate your result with some concrete task sets. (6 points)

Contribution by group member 1 (name, %): ....

Contribution by group member 2 (name, %): ....

PROBLEM 2

This problem concerns the p-fair scheduling algorithm PF presented in [Baruah *et al.*]. Note that the PF algorithm assumes a work-conserving scheduler.

Consider the following task set with three tasks (using the notation in [Baruah *et al.*]):

x	τ_1	τ_2	τ_3
$x.e$	4	4	4
$x.p$	6	6	6

Schedule the task set on $m = 2$ processors using the algorithm PF during the time interval [0,6). Assume that all tasks arrive at time $t = 0$. The algorithm PF permits arbitrary tie-breaking among tasks with equal *characteristic substring*. In this problem, it is assumed that such ties are broken in favor of the task with the lowest index, that is, task τ_1 is given priority over τ_2 and τ_2 is given priority over τ_3.

Show how you achieve your p-fair schedule by including in your solution a detailed step-by-step execution of the PF algorithm, using the format in Table 2 on p. 615 in [Baruah *et al.*]. Also present a timing diagram of the resulting schedule for the time interval [0,6). (8 points)

Contribution by group member 1 (name, %): ....

Contribution by group member 2 (name, %): ....

Reference: S. K. Baruah, N. K. Cohen, C. G. Plaxton and D. A. Varvel: "Proportionate Progress: A Notion of Fairness in Resource Allocation"

PROBLEM 3

This problem concerns the uniprocessor scheduling algorithm presented in [Xu and Parnas].

Consider the following task set with ten tasks (using the notation in [Xu and Parnas]):

$p[i]$	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	T_9	T_{10}
$r[i]$	0	0	12	16	20	32	40	40	40	40
$c[i]$	2	3	6	4	4	1	5	2	1	2
$d[i]$	10	20	26	30	25	35	50	60	70	80

Note that $r[i]$ and $d[i]$ correspond to offset and absolute deadline, respectively, in lecture notes terminology.

Also, consider the following execution constraints on the given task set:

T_1 PRECEDES T_2
T_4 EXCLUDES T_5
T_7 PRECEDES T_8
T_8 PRECEDES T_9
T_9 PRECEDES T_{10}

For the given task set and associated execution constraints, apply the algorithm in [Xu and Parnas] to find the optimal schedule (in terms of lateness). Clearly describe each step in the algorithm, including how the valid initial solution is generated, how the successor (child) nodes in the search tree are generated, and how the algorithm detects that the optimal solution has been found. Timing diagrams should be presented for all schedules (initial, partial and optimal) generated as part of the algorithm. (8 points)

Contribution by group member 1 (name, %): ....

Contribution by group member 2 (name, %): ....

Reference: J. Xu and D. L. Parnas: "Scheduling Processes with Release Times, Deadlines, Precedence, and Exclusion Constraints"

PROBLEM 4

Consider a task set Γ where each task $\tau_i \in \Gamma$ is characterized by (C_i, D_i, T_i) such that $C_i \leq D_i \leq T_i$ (using the notation from the lecture notes). All tasks arrive at time $t = 0$. Assume that Γ is *feasible* using deadline-monotonic (DM) preemptive scheduling on a uniprocessor.

Now consider a new schedulability testing approach, called `OPA_with_RTT`, which works as follows: Audsley's *Optimal Priority Assignment* (OPA) algorithm (see lecture notes) is applied with the exact response-time test for uniprocessor preemptive static-priority scheduling (see lecture notes) to determine the static-priority ordering of the tasks in a task set.

Can you guarantee that `OPA_with_RTT` will generate a priority assignment that makes Γ schedulable? Why or why not? Motivate your answer! (4 points)

Contribution by group member 1 (name, %): ....

Contribution by group member 2 (name, %): ....

PROBLEM 5

The *Slack Stealing* algorithm is without a doubt the most promising method to schedule aperiodic tasks in a system with a static periodic task workload using rate-monotonic (RM) scheduling on a uniprocessor. Unlike techniques that rely on a dedicated server task, the slack stealer is a run-time mechanisms that attempts to find time for servicing aperiodic tasks by delaying the execution of periodic tasks without causing their deadlines to be missed.

To that end, the slack stealer is guided by the following rules:

- It may not violate the priority ordering (that is, the RM policy) among the periodic tasks, in case it is necessary to delay their task executions.
- It may not modify the original arrival pattern of a task, as a means for delaying its execution.
- It should have the run-time behaviour of a preemptive work-conserving scheduler.

Consider the following task set with three tasks (using the notation from the lecture notes), where all tasks arrive at time $t = 0$ and the relative deadline of each task is equal to its period:

	τ_1	τ_2	τ_3
C_i	1	1	1
T_i	3	4	6

Now, assume that that two aperiodic requests occur during the first hyperperiod [0,12) of the periodic tasks: the first request J_1, with $C_{J_1} = 1$, arrives at time $t = 2$, while the second request J_2, with $C_{J_2} = 1$, arrives at time $t = 3$. Note: any aperiodic task that has arrived at time t is assumed to be known to the run-time slack-stealing mechanism for any scheduling decision it takes at time t.

a) Using a timing diagram, show the schedule for the two aperiodic requests when an *myopic* slack stealer is used, that is, one that has no *a priori* knowledge of the requests. This means that the slack stealer cannot begin planning for the handling of an aperiodic task (that is, delaying the execution of periodic tasks) until that particular aperiodic task has actually arrived. (3 points)

Contribution by group member 1 (name, %): ....

Contribution by group member 2 (name, %): ....

b) Using a timing diagram, show the best achievable schedule (in terms of response time) for the two aperiodic requests when a *clairvoyant* slack stealer is used, that is, one that has complete *a priori* knowledge (arrival times and computational demands) of the requests. (3 points)

Contribution by group member 1 (name, %): ....

Contribution by group member 2 (name, %): ....

c) Determine, by calculating and analyzing the response times of the aperiodic requests obtained in sub-problems a) and b), the optimality properties of the slack stealing algorithm. (2 points)

Contribution by group member 1 (name, %): ....

Contribution by group member 2 (name, %): ....

PROBLEM 6

Partitioned multiprocessor scheduling is based on some kind of bin-packing heuristic for assigning the tasks to m processors. During task assignment, whether a task with Rate-Monotonic (RM) priority can be feasibly assigned (i.e. the task meets its deadline) on a particular processor can be determined using the Liu and Layland's Sufficient Feasibility Condition (LLSFC) for uniprocessor RM scheduling algorithm. The LLSFC is given as follows: *if* $U \leq n\,(2^{1/n}-1)$*, then all the n tasks having total utilization U are RM schedulable on one processor.*

Consider four periodic tasks τ_1, τ_2, τ_3 and τ_4 with RM priority to be partitioned on a multiprocessor platform having three processors μ_1, μ_2 and μ_3. It should be assumed that a preemptive work-conserving scheduler is used on each processor.

All tasks arrive at time $t = 0$ and the relative deadline of each task is equal to its period. The worst-case execution time (C_i) and period (T_i) of task τ_i is given in the table below:

	τ_1	τ_2	τ_3	τ_4
C_i	75.0	30.0	21.0	27.5
T_i	150.0	50.0	41.0	31.0

a) Explain why the task set given is not schedulable (or cannot be successfully partitioned) using LLSFC and Rate-Monotonic First-Fit (RMFF) bin-packing heuristic on three processors. (1 point)

Contribution by group member 1 (name, %): ....
Contribution by group member 2 (name, %): ....

Task Splitting partitioned scheduling is a variation of pure partitioned scheduling in which some tasks, called *split tasks*, are allowed to migrate their execution, that is partially completed on one processor, to another processor. In other words, the execution of a split task occurs across multiple processors. The set of partial executions of a split task across multiple processors can be considered as executions of different *subtasks* of the split task (i.e. the accumulated execution of all the subtasks has a behaviour that corresponds, from a software functionality point of view, to that of the original task). The challenge is to determine which task/tasks of a task set with RM priority should be selected for task splitting during task assignment to processors using LLSFC.

Subtasks emanating from the same split task are restricted by the following rules:

- They must execute in sequence, in order to preserve the functional behaviour of the split task.
- They must inherit the period of the split task, in order to preserve the original RM priority ordering among the tasks in the task set.

b) Using the task splitting approach and LLSFC, derive *two* alternative task allocations (partitions) of the given task set so that the task set is RM schedulable on three processors. The alternative allocations may not use the same set of split tasks. For each alternative, state the worst-case execution time for each of the subtasks of each split task and show analytically that all tasks in the task set are RM schedulable for the chosen task splitting solution. Also describe how it is ensured that the subtasks of a split task are not running in parallel across multiple processors. (6 points)

Contribution by group member 1 (name, %): ....
Contribution by group member 2 (name, %): ....

c) Compare the two alternatives in subproblem b) in terms of overhead (e.g. migration, preemption, and other costs) related to task splitting. (1 point)

Contribution by group member 1 (name, %): ....
Contribution by group member 2 (name, %): ....

PROBLEM 7

Consider four hard real-time periodic tasks τ_1, τ_2, τ_3 and τ_4 to be partitioned using the Rate Monotonic First-Fit (RMFF) algorithm on two processors, μ_1 and μ_2. It should be assumed that a preemptive work-conserving scheduler is used on each processor.

As the tasks are periodic, there will be one invocation of task τ_i in each period T_i. The k^{th} invocation (i.e., an instance) of task τ_i is denoted by τ_i^k. All tasks arrive at time $t = 0$ and the relative deadline of each task is equal to its period. The worst-case execution time (C_i) and periods (T_i) of task τ_i are as follows in Table 1:

Table 1: WCET and periods for four tasks

τ_i	C_i	T_i
τ_1	2	12
τ_2	1	4
τ_3	1	8
τ_4	1	3

Duplication and comparison is a well known technique for error detection in fault tolerant systems. Using this technique, each invocation τ_i^k of task τ_i is executed twice, known as the two *primary copies* of τ_i^k. After the two primary copies of τ_i^k finish execution, a comparison of the results of the two primary copies is made. If the two results match, there is no error and we accept the result of the execution. Otherwise, if the two results do not match, an error is detected (and in this case, we do not know which execution is faulty). For this assignment, assume that comparison and error detection time (overhead) is negligible (or, already added in the WCET of the corresponding task) and hence such overhead can be ignored.
Considering the duplication and comparison paradigm, each invocation of task τ_i in Table 1 is executed twice. In other words, each task's execution time is doubled (due to duplicated execution) but the period remains the same. For example, the execution time of all invocations of task τ_1 is now ($2 \times C_1 =$) 4 while the period remains the same, that is, equal to $T_1 = 12$. Consider RMFF scheduling.

a) Can you give any guarantee that all tasks will meet their deadlines if you want each task to execute twice before assigning any task to any processor? Why or why not? (1 point)

Contribution by group member 1 (name, %): ....
Contribution by group member 2 (name, %): ....

Error recovery is the next step after error detection in a fault-tolerant schedule. For this assignment, consider that there is at most ***one*** error within the least-common-multiple of the periods of the tasks. To recover from an error, we decide to run the same instance τ_i^k a *third time* when an error is detected after the two primary copies finish execution. Now we can recover from error by taking a majority vote (2 out of 3) on the results of the three executions. However, running the third copy for some invocations of the task may not always be possible without missing either its own deadline or the deadline of some other task. For the following questions, consider RMFF scheduling in which a *necessary and sufficient* test for uniprocessor is used during task assignment.

b) Generate (draw) the schedule of the tasks on the two processors μ_1 and μ_2 using the RMFF algorithm, with the additional requirement that each task is executed twice. Note that a partitioned approach, such as RMFF, requires all invocations of a task (including any re-execution of that task) to reside on the same processor. Hint: Make sure that, after partitioning, the schedule on each processor has some available slack (that is, no processor should be 100% utilized). (2 points)

Contribution by group member 1 (name, %): ....
Contribution by group member 2 (name, %): ....

c) Identify all such instances τ_i^k of task τ_i from the schedule in subproblem b) for which it applies that, when an error in instance τ_i^k is detected by the duplication and comparison mechanism, then it is not possible to run its third copy without missing either its own deadline or the deadline of some other task. Show your reasoning. (3 points)

Contribution by group member 1 (name, %): ....

Contribution by group member 2 (name, %): ....

Another option for error recovery is that, once an error is detected, a ***recovery routine*** is executed to recover from the error. Consider that the recovery routine of a task has the same priority as the task. Note that the recovery routine must finish execution before the deadline of the instance once an error is detected for that instance.

For subproblems d) and e) below consider at most one error can occur in the RMFF schedule derived in subproblem b) and assume that the error can occur in any task on any processor.

d) What is the upper bound on the execution time of the recovery routine when you do not know in which task the fault may occur during execution? Show your reasoning. (2 points)

Contribution by group member 1 (name, %): ....

Contribution by group member 2 (name, %): ....

e) What is the upper bound on the execution time of the recovery routine when you know that the fault could only occur in tasks executing on processor μ_2? Show your reasoning. (1 point)

Contribution by group member 1 (name, %): ....

Contribution by group member 2 (name, %): ....

There are other ***error detection mechanisms*** in hardware (i.e. invalid memory access, invalid op-code detection, watchdogs) and software (i.e. executable assertions, consistency check) that can detect an error in an instance of a task at an early stage, that is, before the instance finishes both of its primary executions. If an error is detected early, the execution of the recovery copy can be completed before the task's deadline. For subproblem f) below assume that an error can also be detected based on some already build-in error detection mechanism in hardware.

f) For all the instances τ_i^k identified in subproblem c), find the *latest* time at or before which the error must be detected so that the recovery copy of the instance as re-execution could complete without causing any deadline to be missed. Show your reasoning. (3 points)

Contribution by group member 1 (name, %): ....

Contribution by group member 2 (name, %): ....

PROBLEM 8

Read the three highlighted articles listed below, and be prepared during the oral examination to answer questions regarding their technical contents (as specified for each article).

[**Jeffay *et al.***] Study particularly how the transformation from 3-PARTITION is used for proving strong NP-completeness (Theorem 5.2). (2 points)

Reference: K. Jeffay, D. F. Stanat and C. U. Martel: "On Non-Preemptive Scheduling of Periodic and Sporadic Tasks"

[**Davis and Burns**] Study particularly how the strategy with a 'problem window' is used for deriving a sufficient schedulability test (Section 3). (2 points)

Reference: R. Davis and A. Burns: "Improved Priority Assignment for Global Fixed Priority Pre-emptive Scheduling in Multiprocessor Real-Time Systems"

[**Davis *et al.***] Study particularly how this analysis rectifies a serious flaw in an earlier wide-spread analysis for CAN (Section 3). (2 points)

Reference: R. Davis *et al.*: "Controller Area Network (CAN) Schedulability Analysis: Refuted, Revisited and Revised"
