Assignment on Ada

Course EI5064: Real-Time Programming Languages (RTPL)

Technische Universität München
Institute for Real-Time Computer Systems (RCS)
Prof. Dr. sc. Samarjit Chakraborty

December 16, 2015

Name: _____
Matriculation Number: _____
Other group members: _____

This assignment includes 6 questions. The total number of points is 80. In case you solve the exercise in a group, list all group members in the field above.

1 A Certifiable Process for Post-Exam Reviews

Your task is to re-implement the *post-exam process* from your last homework in Ada. In particular, you shall use the *Ravenscar* profile, to obtain an analyzable program which is free of deadlocks and race conditions.

Situation:
The review process works as described in the previous assignment. However, we do not model explicitly that TA or students are changing rooms. That is, if the TA needs to check the waiting room, she can do so by calling a function or using a signalling mechanism. Similarly, the students do not explicitly go to the office and wake up the TA. Instead, the TA shall wake up automatically when a student takes a seat. Since Ada falls into the scheduled paradigm, these actions – even if not explicitly modeled – will automatically take a strictly positive processing time. In particular, this means that it takes time for the students to wake up the TA, for the TA to get a student into review, etc.

Start with the implementation that is posted on Moodle along with these instructions. You find the following files:

- `main.adb`: an empty main procedure, which only references the packages described in the following. No changes required here.
- `ta.ads/adb`: implements a package which creates one task for the TA. Needs to be completed.
- `students.ads/adb`: implements a package which creates one student task. Needs to be completed.
- `waiting_room.ads/adb`: an empty package which shall be used to implement the waiting process. Needs to be completed.
- `review_settings.ads`: defines constants for the program. Can be extended if necessary.

In both the student and the TA task there is call to the procedure `delay_for`. This procedure simulates some processing delay, keeping the caller's process in a running state, and returning only after the specified amount of time. Note that `delay until` cannot be used for this purpose, since this would violate Ravenscar requirements (single release point for tasks). In this assignment there is no need to add further calls to `delay_for`, but the existing ones must be preserved.
Sleeping and Waiting states: We do not explicitly keep track of the sleeping (TA) and waiting (students) states. Instead, it is assumed that the TA is sleeping when its process is blocked, and similarly that the students are waiting when their respective processes are blocked. Processes are blocking when they are sitting in an entry queue or calling `delay until`.

1. (a) (5 points) Extend the task `task_stud`, such that it becomes a periodically (time-triggered) task according to the Ravenscar profile. The period shall be `STUD_PERIOD`.
   (b) (5 points) Extend the task `task_TA`, such that it becomes a sporadic (event-triggered) task according to the Ravenscar profile.
   (c) (10 points) Turn the task `task_stud` into a type, and make three instances of this task, which will represent three student streams. To allow distinguishing between the streams later on, also modify the new task type, such that it accepts a discriminant of type `Students_Range`, and assign the discriminant's value to the variable `id` declared in the task's body (currently initialized to constant zero).
   (d) (25 points) Implement a synchronization mechanism between the TA task and the three student tasks, which permits that students can queue up, and also that the TA

can sleep when nobody is waiting. One way to do this, is to model the seats in the waiting room as shared data. If you make use of protected objects, then make sure that every `entry` has at most one caller, and that entries do not call other entries (required by Ravenscar profile because of `Max_Entry_Queue_Length => 1`).

Important: When running a Ravenscar program on a machine that has no real-time operating system (for example, any standard Ubuntu or Windows), then **the program behaves incorrectly**. The reason is, that an ordinary OS does not follow the task model which Ravenscar requires (FIFO scheduling within priorities and immediate priority ceiling). There are two solutions for this:

1. Get an RTOS or install RT patches, see slides of first Ada lecture, or
2. turn off the Ravenscar profile, clean and re-compile and then run your program. That way, the Ada compiler translates your program differently, such that it is larger, but works correctly without an RTOS. Don't forget to turn it back on for your development.

Hint: Despite the use of the Ravenscar profile, deadlocks are still possible if the software design is bad. As a rule of thumb, if any combination of two or more shared data items could lead to a deadlock (e.g., the variable indicating that the TA sleeps and the variable indicating that a student is waiting to be called in), then these data items should only be changed under mutual exclusion.

Absence of Deadlock and Race Condition

For certification, your program shall be free of deadlock (here: that the TA is sleeping forever and students are waiting at the same time) and race conditions (here: that two students sit on the same chair or go into review at the same time). However, this time we do not use a tool to find a proof.

2. (10 points) Show that your program fulfills these criteria *by construction*, i.e., explain due to which constructs (their semantics) of Ada it is prevented to run into a deadlock or a race condition.

3. (10 points) Explain how the following case is handled by your program, considering that preemption is possible on a real-time system: A student arrives and observes that the TA is not available, and starts to sit down on a chair in the waiting room. While he takes a seat, he is preempted by the TA's process. The TA finishes the review and goes to check the waiting room. Since there is no one there (the student has not finished to take the seat, yet), the TA goes back to her office and sleeps. The student's process resumes and finishes sitting down. The TA is now waiting for a student and the student is waiting for the TA.

Starvation

4. (10 points) Extend your program, such that there is no *starvation*, i.e., every student eventually gets processed.

Minimum Separation

5. (5 points) Change your implementation of the sporadic task `task_TA`, such that a *minimum inter-arrival time/minimum separation time* of `TA_MINT` can be guaranteed.

Bonus Task

6. (10 points) Compute the shortest possible relative deadline for the students tasks, for which they are schedulable on a mono-processor system with the policy `FIFO_within_priorities`, under the following assumptions:

   - there are no other processes apart from those of your program,
   - context switches take no time,
   - the WCET of one student task is `TIME_REVIEW`
   - the WCET of the TA task is `TIME_REVIEW` + `TIME_WALK`
   - the TA's task has a higher priority than the students' tasks
   - period of the student tasks as given is `STUD_PERIOD`
   - the TA has a minimum separation of `TA_MINT` as requested in 5.

   Hint: The WCET does not consider interruptions due to context switches. A context switch also occurs due to preemption by a higher-priority task or if a task is blocking in an entry queue, or if a task has to wait for a `procedure` or `function` call on a protected object because some other task is currently accessing it.

Helpful Literature

- *Guide for the use of the Ada Ravenscar Profile in high integrity systems*, A. Burns and B. Dobbing and T. Vardanega, 2003, online at `http://www.sigada.org/ada_letters/jun2004/ravenscar_article.pdf`.
- *Lecture 10 of Real-Time Programming Languages*, TU München RCS, 2015.
- *Car Blinker Example*, L. Berger, 2015, online at Moodle.

Submit your final Ada program (complete project) at Moodle, as well as **a brief report** informally describing the solution and answering the above questions.