KTH Information and
Communication Technology

IL2206 EMBEDDED SYSTEMS

Laboratory 1 : Concurrent and Real-Time Software Development in Ada

1 Objectives

The programming language Ada has been developed for embedded and real-time systems. Concurrency is supported directly by the language. Ada offers powerful communication mechanisms, like rendezvous and the concept of protected object. Support for real-time systems is provided by the real-time annex. The objective of the laboratory is to introduce the student to Ada 2005, the current standard of the Ada language[1], and its features for concurrency and real-time. For more information on Ada consult the KTH library, which has many books on the Ada programming language.

2 Preparation Tasks

Read the entire laboratory manual in detail before you start with the preparation tasks. Complete the preparation tasks before your lab session in order to be allowed to start the laboratory exercises.

It is very important that students are well-prepared for the labs, since both lab rooms and assistants are expensive and limited resources, which shall be used efficiently. The laboratory will be conducted by groups of two students. However, each student has to understand the developed source code and the preparation tasks. Course assistants will check, if students are well-prepared. Students, who are not well-prepared for the laboratory, have no right to get help from the assistants during the lab sessions!

Whenever you have completed a task of the laboratory, mark the task as completed by putting a cross into the corresponding circle.

Note: All program code shall be well-structured and well-documented. The language used for documentation is English.

[1]Recently the new standard Ada 2012 has been released, but it is still very new, so compilers might not support all features of the new standard.

2.1 Installation of `gnat`

For this laboratory the development tool `gnat` (GNU Ada) will be used. GNAT supports the full real-time annex of Ada 2005 and is part of the `gcc` tool suite. GNAT is available under UNIX and thus there should be no problem to install it, if you use any Linux distribution.

We strongly recommend the use of the virtual machine that is provided by KTH, on which GNAT is already installed. If you want to use a native Linux installation, you need to make sure that you use only **one processor core** when doing this laboratory. If you use Ubuntu you can install `gnat` with the command `sudo apt-get install gnat`. KTH will only provide support for the installation on the virtual machine, and cannot provide any support for own Ada-installations on Windows, Macintosh or Linux.

◯ 2.1 completed

2.2 Cyclic Executive

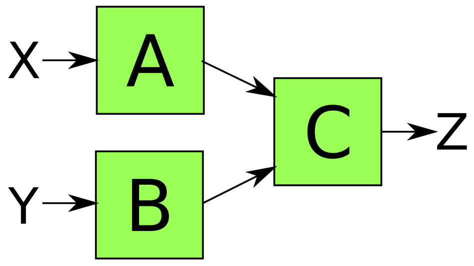

Figure 1: Model of a real-time system

Figure 1 shows a model of a real-time application. The system consumes periodically the inputs `X` and `Y` and produces the output `Z`. The inputs `X` and `Y` are not updated simultaneously, but input `Y` is delayed 500ms compared with input `X`. Also the subsystems A, B and `C` have different execution times ($e_A \approx 100\text{ms}$, $e_B \approx 200\text{ms}$, $e_C \approx 200\text{ms}$). Exact details can be derived from the source files `systems_function_package.ads` and `systems_function_package.adb` that are available on the course web page.

Your task is to develop a cyclic scheduler for the control of the system of Figure 1. The scheduler must respect the timing of the input and the subsystems' execution times. Start with the skeleton program `cyclic_scheduler.adb` that is available on the course web page.

1. Sketch a cyclic schedule for a correct functionality of the application.
2. Complete the code for the task `Scheduler`, so that it correctly processes the inputs `X` and `Y` and correctly implements the function of the system. Whenever a system function (A, B or C has executed, print a message using the following template

   ```
   Put(Duration'Image(To_Duration(A_Time_Span_Variable)));
   Put_Line(": X executed");
   ```

   Also output the final result (`Z`).

◯ 2.2 completed

2.3 Producer-Consumer Problem

The program `producerconsumer.adb` implements a non-reliable implementation of the producer-consumer problem, where data is likely be lost. In the following, you will use three different communication mechanisms to achieve a reliable implementation of the producer-consumer problem.

2.3.1 Semaphore

- The Ada language does not directly provide library functions for a semaphore. However, semaphores can be implemented by means of a protected object. Create a package specification `Semaphore` in the file `Semaphores.ads` and the corresponding package body in the file `Semaphores.adb` that implements a counting semaphore. Skeletons for the package are available on the course page.
- Use the semaphore package for a reliable implementation of the producer-consumer problem. Modify the file `producerconsumer.adb` and save the final code as `producerconsumer_sem.adb`. In order to use the semaphore package it shall be installed in the same directory as `producerconsumer_sem.adb`. It can then be accessed by

  ```
  with Semaphores;
  use Semaphores;
  ```

Note: The file and the main procedure inside it need to have the same name. ◯ 2.3.1 completed

2.3.2 Protected Object

Implement the producer-consumer problem by means of a protected object that implements a buffer of a fixed size. Create a package `Buffer` in the files `buffer.ads` (specification) and `buffer.adb` (body). A skeleton of the file `buffer.ads` is available on the course page. Use the protected object to implement the producer consumer problem and save the final implementation in the file `producerconsumer_prot.adb`. ◯ 2.3.2 completed

2.3.3 Rendezvous

Implement the producer-consumer problem using the rendezvous mechanism. Use the same buffer structure, as in Section 2.3.2, but implement the circular buffer as an own server task. Save your implementation as `producerconsumer_rndvzs.adb`.

◯ 2.3.3 completed

2.4 Real-Time Annex

Note: All programs using the real-time annex must be run as `root`, if you run on a Linux machine. Otherwise the scheduler will not respect the priorities.

2.4.1 Periodic Tasks in Ada

Study the program `periodictask.adb` that implements a few periodic tasks. Before you run the program, try to understand it.

On a specific desktop computer the program results in the following output.

```
> sudo ./periodictasks
  0.023 : 14
  0.023 : 18
  0.023 : 16
  0.023 : 20
  0.047 : 12
  0.047 : 10
  0.273 :   0.273 : 2016

  0.274 : 10
  0.523 : 14
  0.523 : 18
  0.523 :   0.523 : 20
16
  0.547 : 12
  0.547 : 10
  0.773 : 16
  0.773 : 10
  0.774 : 20
```

Answer the following questions:

1. Which task has the highest (lowest) priority?
2. Can you give any information about the architecture of the desktop computer that has been used to run the program?

◯ 2.4.1 completed

2.4.2 Round-Robin Scheduling

Change the scheduling policy to round-robin scheduling by replacing the pragma

```
pragma Task_Dispatching_Policy(FIFO_Within_Priorities);
```

with

```
pragma Priority_Specific_Dispatching(
       Round_Robin_Within_Priorities, 10, 20);
```

Execute the program and compare the results with the results from Section 2.4.1.

◯ 2.4.2 completed

2.4.3 Measurements of Execution Times

The program `executiontimes.adb` measures the execution time of a function that executes a loop n times. Execute this program and use the function `F` with suitable parameters to model the execution time of periodic tasks in the later laboratory tasks.

○ 2.4.3 completed

2.4.4 Rate-Monotonic Scheduling

1. Given is the following set of periodic tasks: $T_1 = (3, 1), T_2 = (4, 1), T_3 = (6, 1)$.

   (a) Calculate the utilization and draw the rate-monotonic schedule for this set of tasks.

   (b) Implement the periodic tasks in Ada, so that the tasks are scheduled rate-monotonically. Replace the normalized times for period and execution time with real times that fit to the execution times measured in Section 2.4.3 (i.e. pick and fix a value for n to determine execution time and adjust the period accordingly). The output of the program shall look like the one presented in Section 2.4.1, i.e., when each task executed its function, print out a time stamp and the task's id. Save the program as `rms.adb`. Execute the program and verify that all deadlines are met.

   Does the program follow the schedule from Subtask 1a? Try to explain possible deviations between the schedule in praxis and the theoretical one.

2. Add now an additional task $T_4 = (9, 2)$. Save the program as `rms2.adb`. Compare the resulting schedule with the one of Subtask 1a. Check, if the program still meets all deadlines.

○ 2.4.4 completed

2.4.5 Watchdog Timer

In order to be able to detect an overloaded system, add both a watchdog timer task and an overload detection task to your program `rms2.adb` and save it as `overloaddetection.adb`. Implement the watchdog timer using rendezvous. The watchdog timer shall issue a warning, if no signal 'OK' is received during one hyperperiod.

1. Run the system with watchdog timer and the task set $\mathbf{T} = \{T_1, T_2, T_3\}$ as described in Section 2.4.4.

2. Run the system with watchdog timer and the task set $\mathbf{T} = \{T_1, T_2, T_3, T_4\}$ as described in Section 2.4.4.

Did you observe a system overload? When did it occur? Explain the results.

○ 2.4.5 completed

2.4.6 Mixed Scheduling

To the program from Task 2.4.4, add now three background tasks that run on a low priority and are scheduled in a round-robin fashion. The tasks shall be repeatedly[2] executed and have a normalized execution time of 1 time unit. Implement this system as `mixedscheduling.adb` using the high-priority task set $\mathbf{T} = \{T_1, T_2, T_3\}$.

In order to enable mixed scheduling, use the following pragmas for the high-priority and the background tasks.

```
pragma Priority_Specific_Dispatching(
                FIFO_Within_Priorities, 2, 30);
pragma Priority_Specific_Dispatching(
                Round_Robin_Within_Priorities, 1, 1);
```

Calculate the time, when the first background task should be executed in theory and compare with the practical result.

◯ 2.4.6 completed

2.4.7 (Optional) Multi-processor execution

If your host machine supports it, increase the number of processors allocated to the VM, for instance to 2. Run `overloaddetection` and `mixedscheduling`. How does this change affect the execution compared to a single-processor run?

Make a rough sketch of the schedule for `overloaddetection`. Does the program follow it? Explain.

3 Laboratory Tasks

3.1 Demonstration of Preparation Tasks

Demonstrate the programs that you have developed in the preparation tasks for the laboratory staff. Be prepared to explain your program in detail.

3.2 Surprise Task

You will be given a 'surprise task', which you need to conduct during the laboratory session. In order to be able to solve the surprise task you need to have a very good understanding of the preparation tasks!

4 Examination

In order to pass the laboratory the student must

- have completed the preparation tasks of Section 2 before the lab session
- have demonstrated the preparation tasks for the laboratory staff and completed the surprise task (Section 3).

[2]Note that repeatedly is not the same as periodically!