IL2206 EMBEDDED SYSTEMS

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

Version 1.2.3

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 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.

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


Documentation

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

Preparation Tasks for this Laboratory

For this laboratory all tasks of Section 3 can be considered as preparation tasks. However, if you have worked seriously and encounter problems, you can still book and visit a laboratory session to discuss with the assistants.

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 2005/2012 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. 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 on the virtual machine, 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, Mac OS, or Linux.

◯ 2.1 completed

Important Notes

- If you use the real-time annex in Ada, the programs need to be run in supervisor mode for the correct timing!
- Ada programs will in general use all cores. If programs shall run on a single core, it has to be enforced by the user. In Linux the user can use the command `sudo taskset -c 0 ./program_name` to enforce execution of a single core.

2.2 Code Skeletons

KTH provides code skeletons for the laboratory, which can be downloaded from the course page in Canvas.

2.3 Modular Types and Attributes in Ada

Ada provides a *modular* integer data type, which can be used to implement a circular buffer. This data type is used in the code skeletons for Section 3.2 provided for the laboratory. In case an addition or other operation yields a result that is larger or smaller than the highest or smallest value that the data type can represent, then “wrap-around semantics” are used. The following examples illustrates the usage of this data type, and also shows how *attributes* like `First` and `Last` can be used to access certain properties of the defined data type. Attributes play an important role in Ada and can be used in different circumstances.

```
with Ada.Text_IO;
use Ada.Text_IO;

procedure Modular_Types is

   type Counter_Value is mod 10;
   package Counter_Value_IO is
      new Ada.Text_IO.Modular_IO (Counter_Value);

   Count : Counter_Value := 5;

begin
   Put("First value of type Counter_Value: ");
   Counter_Value_IO.Put(Counter_Value'First,1);
   Put_Line(" ");
   Put("Last value of type Counter_Value: ");
   Counter_Value_IO.Put(Counter_Value'Last,1);
   Put_Line(" ");
   for I in 1..5 loop
      Count := Count + 6;
      Counter_Value_IO.Put(Count);
      Put_Line("");
   end loop;
end Modular_Types;
```

Execution of the code gives the following output.

```
ada> ./modular_types
First value of type Counter_Value: 0
Last value of type Counter_Value: 9
 1
 7
 3
 9
 5
```

3 Laboratory Tasks

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. Skeletons for the package are available on the course page. Use the package specification `Semaphore` in the file `semaphores.ads` and modify the corresponding package body in the file `semaphores.adb` so that the package implements a counting semaphore.

◯ 3.1 completed

3.2 Producer-Consumer Problem

The producer-consumer problem is a very relevant problem in the design of embedded systems. The problem is illustrated in the Figure 1 and can be for-

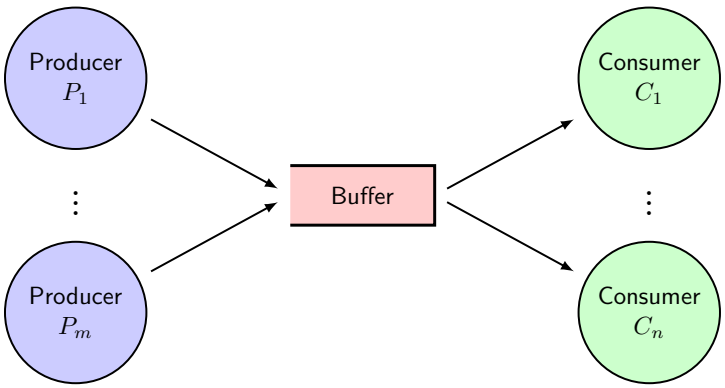

Figure 1: Producer-Consumer Problem

mulated as follows. There are m *producer* and n *consumer* processes, which are connected to a single *buffered communication channel*. Producers write data to the communication channel, consumers read data from the communication channel. For a reliable communication, the following synchronisation properties have to be fulfilled:

- A consumer cannot read a data element from an empty buffer
- A producer cannot write a data element into a full buffer

In the above formulation of the producer-consumer problem, consumers can read data from any sender and are not concerned from which sender the data comes from. When data is read, it is also removed from the buffered channel.

(a) Develop a solution for the producer-consumer problem by means of a *protected object* that uses a buffer of a fixed size. Use the package `Buffer` from the package specification file `buffer.ads` and package body file `buffer.adb`, which together with an initial code skeleton `producerconsumer_prot.adb` for the main program implementation is available on the course web page. Use the protected object to implement the producer consumer problem and save the final implementation in the file `producerconsumer_prot.adb`.

> **Modular Types in Ada**
> The buffer implementation uses modular types. These enable define data types, which allow a 'wrap-around' if the maximum value (or minimum value) is reached and the next value (or previous value) should be calculated. For instance, if a variable `V` is implemented as a modular integer type between 0 and 5, and the current value is 5, then adding 1 to the variable `V` would give the result 0.

Please study the executable example on modular types in Ada, which is available in the Lab1's Canvas page to understand how modular types can be used.

Note: The main procedure and its corresponding source code file need to have the same name. Thus the main procedure needs to have the name `ProducerConsumer_Prot`.

◯ 3.2-a completed

(b) Develop a solution for the producer-consumer problem using the *rendezvous* mechanism. Use the same buffer structure as in Task a, but implement the circular buffer as an own server task. Save your implementation as `producerconsumer_rndvzs.adb`. An initial skeleton for `producerconsumer_rndvzs.adb` can be found on the course web page.

◯ 3.2-b completed

(c) Develop a solution for the producer-consumer problem using your *semaphore* implementation from Task 3.1. Use the same buffer structure as in Task a, but implement the circular buffer as a shared variable. An initial skeleton for `producerconsumer_sem.adb` can be found on the course web page. 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 the following code.

```
1 with Semaphores;
2 use Semaphores;
```

Use *two semaphores* `NotFull` and `NotEmpty`, which shall be used to block a) tasks that want to write to a full buffer, and b) tasks that want to read from an empty buffer, and *another semaphore* `AtomicAccess` to ensure mutual exclusive access to the buffer data structure. Your code shall use the code for the buffer provided in the skeleton `producerconsumer_sem.adb` and shall not be based on a protected object or rendezvous.

Draw also the diagram that illustrates your solution extending the figure above.

◯ 3.2-c completed

3.3 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.

3.3.1 Periodic Tasks in Ada

A real-time Ada program, where six tasks with fixed priorities are scheduled priority-driven, produce the following output on a single core computer.

```
> sudo taskset -c 0 ./periodictasks_analyse_priority
Warm-Up - No task released for  100 Milliseconds
Task 4- Release:  0.200, Completion: 0.374, Response: 0.174, WCRT: 0.174, Next Release:  0.900
Task 5- Release:  0.400, Completion: 0.582, Response: 0.182, WCRT: 0.182, Next Release:  1.000
Task 6- Release:  0.600, Completion: 0.702, Response: 0.102, WCRT: 0.102, Next Release:  1.000
Task 1- Release:  0.800, Completion: 0.946, Response: 0.146, WCRT: 0.146, Next Release:  1.700
Task 2- Release:  1.000, Completion: 1.089, Response: 0.089, WCRT: 0.089, Next Release:  1.500
```

```
Task 6- Release:  1.000, Completion: 1.179, Response: 0.179, WCRT: 0.179, Next Release:  1.400
Task 3- Release:  1.200, Completion: 1.289, Response: 0.089, WCRT: 0.089, Next Release:  2.000
Task 5- Release:  1.000, Completion: 1.359, Response: 0.358, WCRT: 0.358, Next Release:  1.600
Task 4- Release:  0.900, Completion: 1.396, Response: 0.496, WCRT: 0.496, Next Release:  1.600
Task 2- Release:  1.500, Completion: 1.589, Response: 0.089, WCRT: 0.089, Next Release:  2.000
Task 6- Release:  1.400, Completion: 1.592, Response: 0.192, WCRT: 0.192, Next Release:  1.800
Task 1- Release:  1.700, Completion: 1.788, Response: 0.088, WCRT: 0.146, Next Release:  2.600
Task 6- Release:  1.800, Completion: 1.889, Response: 0.089, WCRT: 0.192, Next Release:  2.200
Task 5- Release:  1.600, Completion: 1.897, Response: 0.297, WCRT: 0.358, Next Release:  2.200
Task 4- Release:  1.600, Completion: 1.986, Response: 0.386, WCRT: 0.496, Next Release:  2.300
Task 3- Release:  2.000, Completion: 2.096, Response: 0.096, WCRT: 0.096, Next Release:  2.800
Task 2- Release:  2.000, Completion: 2.185, Response: 0.185, WCRT: 0.185, Next Release:  2.500
Task 6- Release:  2.200, Completion: 2.322, Response: 0.122, WCRT: 0.192, Next Release:  2.600
Task 5- Release:  2.200, Completion: 2.413, Response: 0.213, WCRT: 0.358, Next Release:  2.800
Task 2- Release:  2.500, Completion: 2.588, Response: 0.088, WCRT: 0.185, Next Release:  3.000
Task 4- Release:  2.300, Completion: 2.591, Response: 0.291, WCRT: 0.496, Next Release:  3.000
Task 1- Release:  2.600, Completion: 2.729, Response: 0.129, WCRT: 0.146, Next Release:  3.500
Task 3- Release:  2.800, Completion: 2.889, Response: 0.089, WCRT: 0.096, Next Release:  3.600
Task 6- Release:  2.600, Completion: 2.908, Response: 0.308, WCRT: 0.308, Next Release:  3.000
Task 5- Release:  2.800, Completion: 2.998, Response: 0.198, WCRT: 0.358, Next Release:  3.400
Task 2- Release:  3.000, Completion: 3.090, Response: 0.090, WCRT: 0.185, Next Release:  3.500
Task 6- Release:  3.000, Completion: 3.180, Response: 0.180, WCRT: 0.308, Next Release:  3.400
Task 4- Release:  3.000, Completion: 3.270, Response: 0.270, WCRT: 0.496, Next Release:  3.700
Task 2- Release:  3.500, Completion: 3.589, Response: 0.089, WCRT: 0.185, Next Release:  4.000
Task 3- Release:  3.600, Completion: 3.689, Response: 0.089, WCRT: 0.096, Next Release:  4.400
Task 1- Release:  3.500, Completion: 3.768, Response: 0.268, WCRT: 0.268, Next Release:  4.400
Task 6- Release:  3.400, Completion: 3.795, Response: 0.395, WCRT: 0.395, Next Release:  3.800
Task 6- Release:  3.800, Completion: 3.889, Response: 0.089, WCRT: 0.395, Next Release:  4.200
Task 5- Release:  3.400, Completion: 3.975, Response: 0.575, WCRT: 0.575, Next Release:  4.000
Task 2- Release:  4.000, Completion: 4.089, Response: 0.089, WCRT: 0.185, Next Release:  4.500
Task 5- Release:  4.000, Completion: 4.179, Response: 0.179, WCRT: 0.575, Next Release:  4.600
Task 6- Release:  4.200, Completion: 4.289, Response: 0.089, WCRT: 0.395, Next Release:  4.600
Task 4- Release:  3.700, Completion: 4.334, Response: 0.634, WCRT: 0.634, Next Release:  4.400
Task 3- Release:  4.400, Completion: 4.547, Response: 0.147, WCRT: 0.147, Next Release:  5.200
Task 2- Release:  4.500, Completion: 4.636, Response: 0.136, WCRT: 0.185, Next Release:  5.000
Task 1- Release:  4.400, Completion: 4.726, Response: 0.326, WCRT: 0.326, Next Release:  5.300
Task 6- Release:  4.600, Completion: 4.817, Response: 0.217, WCRT: 0.395, Next Release:  5.000
Task 5- Release:  4.600, Completion: 4.907, Response: 0.307, WCRT: 0.575, Next Release:  5.200
Task 4- Release:  4.400, Completion: 4.997, Response: 0.597, WCRT: 0.634, Next Release:  5.100
Task 2- Release:  5.000, Completion: 5.090, Response: 0.089, WCRT: 0.185, Next Release:  5.500
Task 6- Release:  5.000, Completion: 5.179, Response: 0.179, WCRT: 0.395, Next Release:  5.400
Task 3- Release:  5.200, Completion: 5.289, Response: 0.089, WCRT: 0.147, Next Release:  6.000
Task 1- Release:  5.300, Completion: 5.390, Response: 0.090, WCRT: 0.326, Next Release:  6.200
Task 6- Release:  5.400, Completion: 5.489, Response: 0.089, WCRT: 0.395, Next Release:  5.800
Task 2- Release:  5.500, Completion: 5.590, Response: 0.090, WCRT: 0.185, Next Release:  6.000
Task 5- Release:  5.200, Completion: 5.648, Response: 0.448, WCRT: 0.575, Next Release:  5.800
Task 4- Release:  5.100, Completion: 5.718, Response: 0.617, WCRT: 0.634, Next Release:  5.800
Task 3- Release:  6.000, Completion: 6.089, Response: 0.089, WCRT: 0.147, Next Release:  6.800
Task 2- Release:  6.000, Completion: 6.179, Response: 0.179, WCRT: 0.185, Next Release:  6.500
Task 6- Release:  5.800, Completion: 6.182, Response: 0.382, WCRT: 0.395, Next Release:  6.200
Task 1- Release:  6.200, Completion: 6.289, Response: 0.089, WCRT: 0.326, Next Release:  7.100
Task 6- Release:  6.200, Completion: 6.379, Response: 0.179, WCRT: 0.395, Next Release:  6.600
Task 5- Release:  5.800, Completion: 6.452, Response: 0.652, WCRT: 0.652, Next Release:  6.400
Task 2- Release:  6.500, Completion: 6.589, Response: 0.089, WCRT: 0.185, Next Release:  7.000
Task 6- Release:  6.600, Completion: 6.689, Response: 0.089, WCRT: 0.395, Next Release:  7.000
Task 5- Release:  6.400, Completion: 6.720, Response: 0.320, WCRT: 0.652, Next Release:  7.000
Task 3- Release:  6.800, Completion: 6.889, Response: 0.089, WCRT: 0.147, Next Release:  7.600
Task 4- Release:  5.800, Completion: 6.898, Response: 1.098, WCRT: 1.098, Next Release:  6.500
Task 4- Release:  6.500, Completion: 6.988, Response: 0.488, WCRT: 1.098, Next Release:  7.200
Task 2- Release:  7.000, Completion: 7.130, Response: 0.130, WCRT: 0.185, Next Release:  7.500
Task 1- Release:  7.100, Completion: 7.221, Response: 0.120, WCRT: 0.326, Next Release:  8.000
Task 6- Release:  7.000, Completion: 7.310, Response: 0.310, WCRT: 0.395, Next Release:  7.400
Task 6- Release:  7.400, Completion: 7.489, Response: 0.089, WCRT: 0.395, Next Release:  7.800
Task 5- Release:  7.000, Completion: 7.490, Response: 0.489, WCRT: 0.652, Next Release:  7.600
Task 2- Release:  7.500, Completion: 7.590, Response: 0.090, WCRT: 0.185, Next Release:  8.000
Task 3- Release:  7.600, Completion: 7.690, Response: 0.090, WCRT: 0.147, Next Release:  8.400
Task 5- Release:  7.600, Completion: 7.780, Response: 0.180, WCRT: 0.652, Next Release:  8.200
Task 6- Release:  7.800, Completion: 7.889, Response: 0.089, WCRT: 0.395, Next Release:  8.200
Task 4- Release:  7.200, Completion: 7.939, Response: 0.739, WCRT: 1.098, Next Release:  7.900
Task 2- Release:  8.000, Completion: 8.089, Response: 0.089, WCRT: 0.185, Next Release:  8.500
Task 1- Release:  8.000, Completion: 8.178, Response: 0.178, WCRT: 0.326, Next Release:  8.900
Task 6- Release:  8.200, Completion: 8.289, Response: 0.089, WCRT: 0.395, Next Release:  8.600
Task 5- Release:  8.200, Completion: 8.379, Response: 0.179, WCRT: 0.652, Next Release:  8.800
Task 4- Release:  7.900, Completion: 8.386, Response: 0.486, WCRT: 1.098, Next Release:  8.600
```

Try to order the tasks according to their task priority. Give a motivation how you have arrived at your conclusion. ◯ 3.3.1 completed

3.3.2 Rate-Monotonic Scheduling

Given is the following set of periodic tasks:

$$\Gamma_1 = \{\tau_1(100, 300, 100, 300), \tau_2(100, 400, 100, 400), \tau_3(100, 600, 100, 600)\}$$

where the times are given in milliseconds. A periodic task τ_i is defined as a tuple $\tau_i(\phi_i, T_i, C_i, D_i)$, where ϕ_i denotes the phase, T_i the period, C_i the computation time, and D_i the relative deadline.

1. Calculate the utilisation and draw the rate-monotonic schedule for this set of tasks for one hyperperiod.
2. Given is the skeleton program `periodictasks_priority.adb`. Run the program for some time iterations on a single core[1] and calibrate the program by adjusting the constant `Calibrator`, so that the measured worst case execution time for the running task is close to the given computation time.
3. Implement the periodic task set Γ_1 in Ada, so that the tasks are scheduled rate-monotonically. Build your implementation on the calibrated skeleton program `periodictasks_priority.adb`.
   (a) Save the program as `rms.adb`. Execute the program and validate the expected behaviour.
   (b) Save the output from one simulation in electronic format and provide it as part of your solution.
   (c) Run the program several times for a few hyperperiods. Does the program follow the schedule from Task 1? Try to explain possible deviations between the schedule in reality and the theoretical one.
   (d) (Optional) If possible, run the program `rms.adb` using all cores on your computer[2].
4. Add now an additional task $\tau_4 = (100, 1200, 200, 1200)$.
   (a) Draw the schedule for the program for one hyperperiod.
   (b) Save the program as `rms2.adb`. Run the program several times for a few hyperperiods. Compare the resulting schedule with the one of Task 4a. Explain possible deviations.
   NOTE: In case you see no deadline violations, increase the length of the computation times by adjusting the constant `Calibrator`.
   (c) Save the output from one simulation in electronic format and provide it as part of your solution.

◯ 3.3.2 completed

[1] In Linux you can use the command `sudo taskset -c 0 ./rms` to enforce execution of a single core.

[2] In Linux the cores 0 to 3 will be used, if you use the command `sudo taskset -c 0,1,2,3 ./rms2`. For more information on your cores, run `sudo less /proc/cpuinfo`.

3.3.3 Overload Detection with Watchdog Timer

In order to be able to detect an overloaded system, add both a watchdog timer task and one additional helper task to your program `rms2.adb` and save it as `overloaddetection.adb`.

The watchdog timer and the helper task shall communicate with each other using rendezvous. An overload happens when there is not enough free CPU capacity to run all the user tasks until completion within a hyper-period. Study the principal functionality of a watchdog timer in the lecture notes and design the watchdog timer. Then think about how an additional helper task should be designed, so that the watchdog timer task can detect an overloaded system. In case of an overload, the watchdog timer shall output a warning. Make a sketch of your design and be prepared to explain the functionality of your solution to the course staff. To solve this task, it is important to understand how the fixed-priority scheduling algorithm determines which task should be scheduled and executed, and when these decisions are taken. Thus, you also have to think about which priorities should be given to the watchdog timer task and the helper task.

1. Run the system with overload detection and the task set $\mathbf{\Gamma_1} = \{\tau_1, \tau_2, \tau_3\}$ as described in Section 3.3.2-3.
2. Run the system with overload detection and the task set $\mathbf{\Gamma_2} = \{\tau_1, \tau_2, \tau_3, \tau_4\}$ as described in Section 3.3.2-4. Did you observe a system overload? When did it occur? If not, increase the workload. Explain the results.

◯ 3.3.3 completed

3.3.4 Mixed Scheduling

Add now three background tasks to the program from Task 3.3.2-3, which run on a low priority and are scheduled in a round-robin fashion. The tasks shall be repeatedly[3] executed and each background task has an execution time of 100 milliseconds. Implement this system as `mixedscheduling.adb` using the high-priority task set $\mathbf{\Gamma_1} = \{\tau_1, \tau_2, \tau_3\}$.

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.

◯ 3.3.4 completed

3.3.5 (Optional) Multi-processor execution

If your host machine supports it, increase the number of processors allocated to the VM, for instance to 2. Run the programs `overloaddetection` of

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

Task 3.3.3-2 and `mixedscheduling` of Task 3.3.4. 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?

4 Examination

Demonstrate the programs that you have developed for the laboratory staff during your laboratory session. Be prepared to explain your program in detail. In order to pass the laboratory the student must have completed all tasks of Section 3 and have successfully demonstrated them for the laboratory staff.