Lab Real-time tasks

May 2022

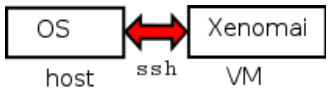
Jacques Lamaison, Janette Cardoso, Pascal Chauvin, Alfonso Mascarenas-G., Alain Hostalier, E. Metral

Goal of this lab: Understand *tasks* (role of priority) and *control mechanisms* (mutex and semaphore) by running a real-time program (written in C language) with 2 tasks.

In this lab, the *real-time control architecture* is already provided: the tasks and synchronization and/or communication (see Fig. 2).

Once compiled, the C code will be executed in a target with the *real-time operation system* Xenomai running[1] in an VM (Virtual Machine) in a host computer (MacOS, Windows or Linux), at ISAE or on your computer. The code is executed using NetBeans, an open source IDE (Integrated development environment).

The remote login to the target is made using a ssh protocol from your host computer through NetBeans.



You will do a short report (to be uploaded on LMS) with the results obtained in the exercises. Refer to the slides for understanding and answering the questions.

Steps for the lab:

All docs are on the LMS page: https://lms.isae.fr/course/view.php?id=1518#section-2

1. You need VirtualBox (VB), Xenomai, and NetBeans installed.
In rooms 07.061 and 062, you need to add the VM Xenomai to VirtualBox (already installed): Open VirtualBox, click **Add** button and browse the vbox file in `D:/VM_HDD/isae3a_dsat/vbox/xenomai-2.3.4/windows-host.vbox`. If you want to install in your computer, see instructions in the LMS page of 1MAE803.
2. Download the file PeriodicTasks.zip (from the 1MAE803LMS page) in folder $home/xenomai803 (see userGuideVB_v*.pdf). Don't unzip this file.
3. Import the zip file as a project in NetBeans: Navigate into the File menu: `File → Import Projet → From ZIP` (see more details in section 1 in GettingStartedNetBeans_v*.pdf).
4. Configure the target Xenomai VM and connect to the target: see section 2 of GettingStartedNetBeans_v*.pdf. Be sure you have a result in Netbeans similar as the one on Fig. 1.

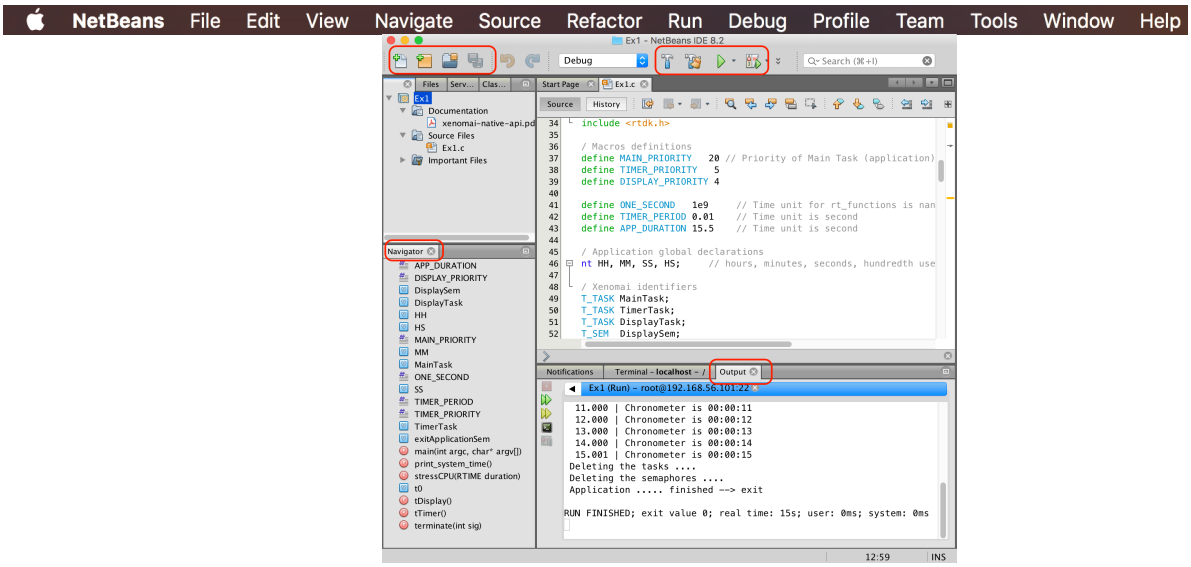


Fig. 1: NetBeans window with project Ex1

[1] In the ISAE machines in room 07.061 and 07.062, the code can be executed in ARM Cortex embedded in the satellite wheel, named `disc-wheelX (x=1..6)`. This target is the same used for the control of the wheel in others courses. There is only 6 targets so please check with your colleagues for using it one at once.

Exercise 1: Getting started

Goal: Getting started with a C code and Xenomai real-time services (commands starting by rt_, for real-time).

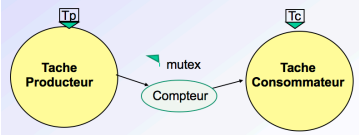


Fig. 2:Architecture and communication between the tasks.

1. Make sure you imported PeriodicTasks.zip in NetBeans. Open file PeriodicTasks.c:
Menu `Project`/, PeriodicTasks/SourceFiles/PeriodicTasks.c. Compare with Fig. 1 and observe the different parameter definitions on tab Navigator (on the left) and the PeriodicTasks.c code. You'll find the task priorities (e.g., PRODUCER_PRIORITY), time values (e.g., PRODUCER _PERIOD), the Xenomai identifiers (e.g., exitApplicationSem, counterMutex) and the functions (e.g., DisplayTaskInfo).

2. Answer these questions in the (short) report. If necessary, read the slides presented in the lecture :
a. Check the name of the tasks, task identifiers `RT_TASK` and semaphore identifier `RT_MUTEX`
b. Check task priorities (PRODUCER_PRIORITY=6 and CONSUMER_PRIORITY=5) and periods of the tasks (PRODUCER_PERIOD, CONSUMER_PERIOD). Are they coherent considering we are using Rate Monotonic?
c. How (or why) the mutex named `CounterMutex` is used here?

3. Observe the following in the code and write down in the report (you can click on the definitions on tab Navigator for localizing it in the C code):
a. All the RTOS services `rt_*,` in particular, the control execution task services `rt_task_*`
b. In main: Creation of the 2 tasks (check the service `rt_task_spawn` and its parameters); creation of the semaphore (`rt_sem_create`) et mutex (`rt_mutex_create`);
c. Take a look on function `forceCPUuse`
d. The body of the task tProducer, its activation period and mutex rt_mutex_acquire and rt_mutex_release.
e. Do the same for task tConsumer.

4. Open the file xenomai-native-api.pdf in folder PeriodicTasks/documentation (in NetBeans, or using a pdf viewer). Find the definition for 2 or 3 services `rt_` used in the C code.

4. Now, run the code: click on the hammer icon, and if the build succeeds, click in green arrow icon.

Wait for the end of execution (APP_DURATION = 10s). You can stop the execution by doing CTRL-C (clicking on the red button does not work (window Output inFig. 1), *but for the report you need to run for 10 seconds*. You must have a trace as the one depicted in Fig. 4. Copy the trace on the NetBeans window and paste in a Exo1StudNames.txt file (StudNames is your name). Observe the results: task response time (wort, best, average). At this point, we don't worry about the critical section in both tasks.

5. Using the C code, or the corresponding State Machine in Fig. 3, "simulate" the behavior (considering the period of the tasks and the priority) "playing" the role of the scheduler, and **make a time diagram**. Indicate in the diagram the state of each task (A,W,Re).

6. Use Cheddar for running a simulation of a (quite) similar scheduling (file `prodConsText` on LMS). Compare the Cheddar time diagram (see Fig. 11) with the one you did by hand. See FAQ 2 and FAQ 3 for using Cheddar.

7. Check the model: a) the scheduler used (menu Edit/Hardware/Core); b) how the tasks are defined (Edit/ Software/ Task).

Exercise 2: Understanding the execution

Observe the first line: `counter= 20 time: 20xxxx.000` in Fig. 4.

Question: Explain the value of counter and time in this instruction.

```
rt_printf("counter= %d time :%8.3f  \n\r",c,(double)((rt_timer_read()-t0)/1000));
```

Hint: Check the period of the tasks; its relative values, etc. Make a *zoom* in your time diagram for seeing when the variable counter is written and read.

Remark: The explanation of how the program stops will be seen in Exercise 5 (Semaphore).

Remark: For a better understanding, in parallel, check the execution of the tasks in the Xenomai VM. For seeing the different information, in the Xenomai VM terminal type:

ls /proc/xenomai/ : for using a French keyboard.
loadkeys fr

cat /proc/xenomai/sched : the tasks with period, etc.
cat /proc/xenomai/stat: occupancy rate at a given time.

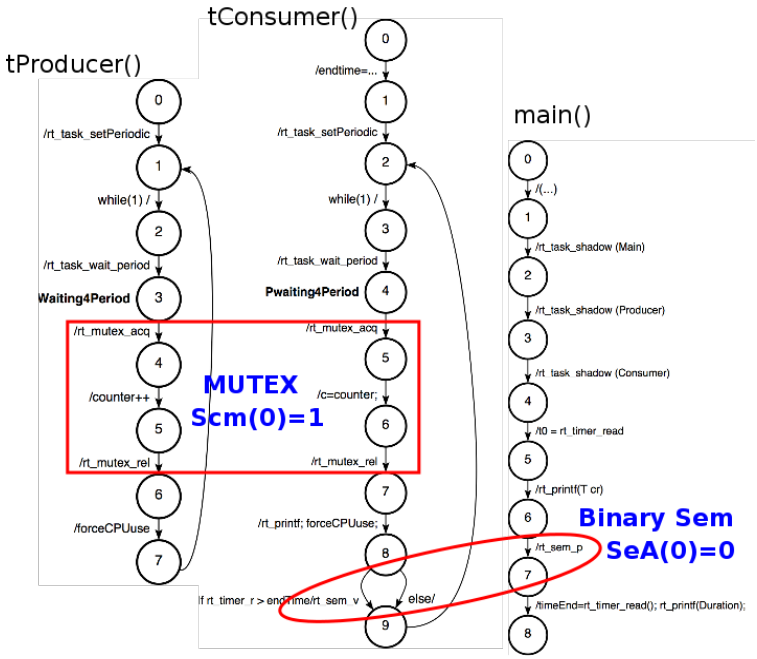


Fig. 3: State machines of tasks: main, tProducer and tConsumer

```
Period Prod: 10ms, Period Cons: 200ms
  Creating the application mutexes ....
   Creating the application tasks ....
   Tasks .. created and launched

  counter= 20 time :201466.000
  counter= 40 time :401103.000
  counter= 60 time :601067.000
  counter= 80 time :801105.000
  counter= 100 time :1001089.000
  counter= 120 time :1201100.000
  (...)
  counter= 940 time :9401151.000
  counter= 960 time :9601146.000
  counter= 980 time :9801078.000
  counter= 1000 time :10001667.000
   total Duration =  10057100 µs  ( 10.057100 s )
  Task name : Producer, Priority : 6, Exectime MP en µs 1066757
  Commutations P/S : 0 , Mode : 300184 , Context switches: 1006
  Task name : Consumer, Priority : 5, Exectime MP en µs 2502489
  Commutations P/S : 0 , Mode : 300184 , Context switches: 302
   Deleting the tasks ....
   Deleting the mutexes ....
   Application ..... finished --> exit

  RUN FINISHED; valeur renvoyée 0 ; real time: 10s; user: 0ms;
  system: 0ms
```

Fig. 4: Prio Producer:6, Prio Consumer:5.

Exercise 3: Change the execution duration of a task

The goal here is show the importance of a good measure of the execution duration of the task. This value change for different execution, and is necessary to have a good value for the WCET (Worst Case Execution Time). This value is important when doing analytical validation, as model checking.

The function `forceCPUuse` is used for modifying the execution duration of a task and see what changes. In this lab, we are not doing any analysis, only simulation. The code cannot be interrupted during the execution of an instruction.

Question 3.1: Now, try CONSUMER_WCET = 150.000 and observe the new values, they must be similar to the ones in Fig. 6. What changed compared to then of the trace in Fig. 5., and why?
Hint: look the total duration and the context switches of tConsumer.

Remark: a **context switch** happens each time the CPU interrupts the execution of a task, or when it starts the execution of a task. An interruption occurs if different cases: is preempted by a task with a bigger priority; the task is waiting for its activation at the next period; the task does not have a semaphore, etc. Remember that only the task that is Active will have its instructions executed.

```
total Duration =  10056489 µs  ( 10.056490 s )
Task name : Producer, Priority : 6, Exectime MP en µs 1132564
Commutations P/S : 0 , Mode : 300184 , Context switches: 999
Task name : Consumer, Priority : 5, Exectime MP en µs 2507180
Commutations P/S : 0 , Mode : 300184 , Context switches: 303
```

Fig. 5: Trace for PRODUCER_CPUuse=1000, CONSUMER_CPUuse =50000

```
total Duration =  10167432 µs  ( 10.167432 s )
Task name : Producer, Priority : 6, Exectime MP en µs 1041157
Commutations P/S : 0 , Mode : 300184 , Context switches: 1015
Task name : Consumer, Priority : 5, Exectime MP en µs 7501324
Commutations P/S : 0 , Mode : 300184 , Context switches: 852
```

Fig. 6: Trace for PRODUCER_CPUuse =1000, CONSUMER_CPUuse =150000

Question 3.2: Draw a time diagram for the case of Fig. 6. Compare with the one you found for Fig. 5 (same as question 1.5).

Exercise 4: Change the priority of tConsumer

You can make a copy PeriodicTasks.c in another folder for having a trace of your changes. Anyway, you always have on the LMS the original file. But does not change the name in Netbeans, neither keep the copy in the same folder.

Change the priority of tConsumer to 7 and observe the trace. It must be similar to the one on Fig. 7.

Question 4.1: Draw a time diagram for each scheduling (Fig. 7 and Fig. 8). **Remark:** This scenario cannot be simulated by Cheddar, because it sets automatically the good priorities based on the period of the tasks.

Question 4.2. Why the counter shows 19, 34, etc., instead of 20, 40 as in Fig. 4?
Hint: Observe the context switches again...

```
counter= 19 time :200425.000
counter= 34 time :400633.000
counter= 50 time :600560.000
(...)
counter= 726 time :9600037.000
counter= 741 time :9800095.000
counter= 756 time :10000072.000
 total Duration =  10051127 µs  ( 10.051127 s )
Task name : Producer, Priority : 6, Exectime MP en µs 788447
Commutations P/S : 0 , Mode : 300184 , Context switches: 758
Task name : Consumer, Priority : 7, Exectime MP en µs 2504732
Commutations P/S : 0 , Mode : 300184 , Context switches: 51
```

Fig. 7: Trace for CONSUMER_CPUuse =50000

```
 counter= 19 time :200075.000
counter= 24 time :400081.000
counter= 29 time :600130.000
(...)
counter= 266 time :9600129.000
counter= 271 time :9800106.000
counter= 276 time :10000129.000
 total Duration =  10151267 µs  ( 10.151267 s )
Task name : Producer, Priority : 6, Exectime MP en µs 289464
Commutations P/S : 0 , Mode : 300184 , Context switches: 278
Task name : Consumer, Priority : 7, Exectime MP en µs 7502962
Commutations P/S : 0 , Mode : 300184 , Context switches: 51
```

Fig. 8: Trace for CONSUMER_CPUuse =150000

Exercise 5: Semaphore

In this example, there is a semaphore used for stopping the execution of the program. The execution of the program stops when the user does a CTRL-C (function terminate) or when the execution duration is greater than APP_DURATION, when the semaphore `exitApplicationSem` is released. Check the code and the State Machine in Fig. 3.

```
t0 = rt_timer_read(); // used by print_system_time() function
rt_sem_p(&exitApplicationSem, TM_INFINITE);
timeEnd = rt_timer_read(); // used by print_system_time() function
```

Observe the parameters `mode` and `icount` of `rt_sem_create` in the Xenomai doc, sec. 4.12.2.3, pg 91.
Question: Which is the initial value of `icount` in the code? Why using this value as an initial value?

Exercise 5: Understanding a critical section

A critical section is protected by a mutex. It's necessary if 2 or more tasks needs to have access to a date (for reading and writing). See the handout about tasks and mutex. See Fig. 9.

For introducing a resource in Cheddar (and so, a critical section with a mutex) we need to multiple all time values by 100:
- Task Consumer: Pc=2000ms, cc=500ms, prioc 5 ;
- Task Producer: Pp=100ms, cp=10ms, priop 7.

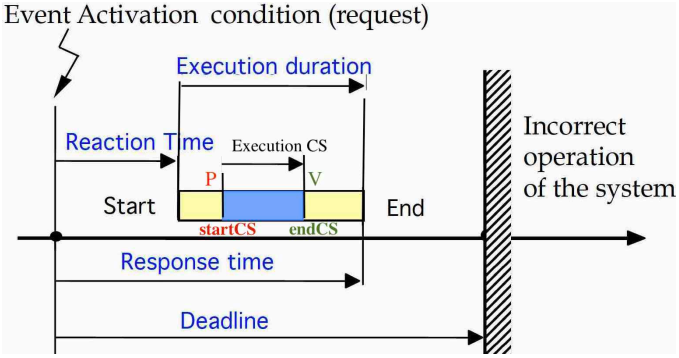


Fig. 9. Considering a critical section (CS) in a task

1. Import the zip file PeriodicTasksCS.zip in NetBeans, and run the model. The code is exactly the same as PeriodicTasks.c, but execution durations where inserted *before*, *during* and *after* the critical sections. Check the new variables *_StartCS and *_durationCS, and the C code related to the critical section.

The values hard-coded in the C code are the following:
- critical section of Consumer code [50,70] (duration 20ms)
- critical section of Producer code [1,8] (duration 7ms).

2. Run the models below, and analyze the results. Are they coherent?

a. model prodCons: no critical section (same model as in prodConsText, but with time values multiplied by 10). Results are similar to the one obtained in Fig. 11, just the times are multiplied by 10.

b. model prodConsCriticalSectionSmall: CSp=[2,8] CSc=[51,70]. In this model we consider the duration of the critical section. Check the model: Edit/Software/Resource. Click on the line in the right panel of the window, you will see the parameters on the bottom left.

Remark: In Cheddar models, a critical section [a,b] is represented in Cheddar as [a+1,b].

3. Now put the following values:
a. In the **C code**: change only the critical section of Consumer: [50,450] (duration 400ms).
Run the C code, and put the results in your report.

b. In **Cheddar:** save the prodConsCriticalSectionSmall as prodConsCriticalSectionBig.

Important remark: the current version of Cheddar (3.2) does not allow to simulate the behavior of Xenomai (concerning a task that misses its deadline). In Xenomai, if a task loses its deadline, it's done. In Cheddar 3.2, if a task T1 with period P is blocked by a task with smaller priority T2 for k*P units of time, Cheddar put T1 in a queue and when T2 finishes, it run T1 k times. In our example it could be a good thing because it will count, but if T1 is acquiring a sensor value and writing it in a memory, it will read k times the same value, and worst, it had missed the previous values. May be one of the values was a threshold value that would set an alarme... BTW, in the version 3.3, expected for July, there will be an option to decide what will happen when a job misses its deadline.

In this **new Cheddar model**, change only the end of task Cons (450 instead of 70). See Fig. 10 for the steps. Sorry, there is 2 more steps: step 1.a (select the gray line 'cons 50 70') and step 3a (delete the gray line 'cons 50 70'). Follow steps in this order: 1 (select gray line on the right), 1a (select gray line on the left), 2 (change value), 3 (Add), 3a (Delete), 4 (Modify on the bottom). Save the model before running.
Run the Cheddar model. What you can observe? Do a capture of your results, put in the report and explain. Check the remark below.

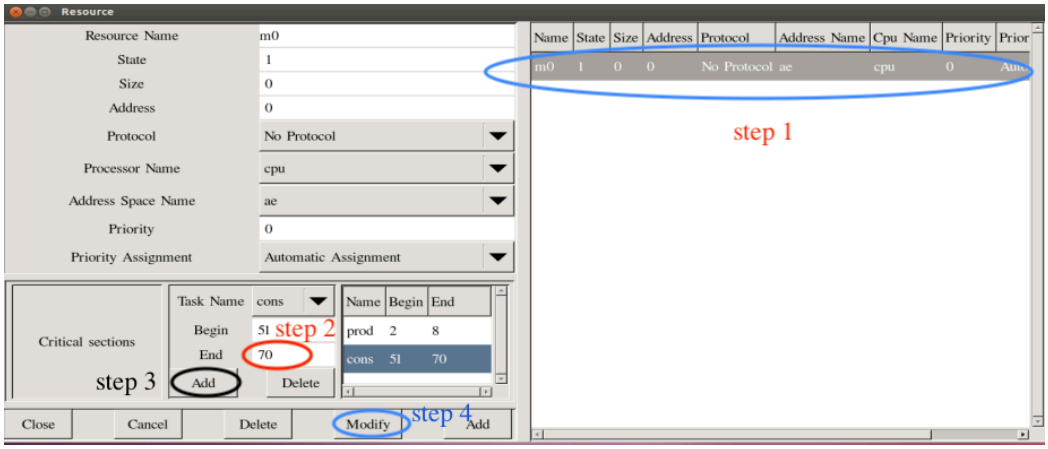


Fig. 10. Changing the end of a critical section in Cheddar

FAQ:

1. Why I have the message " **RUN FAILED**", or nothing appear in the NetBeans terminal?
Answer: One reason is that the execution duration of a task is greater than its period which must be *really* avoided!

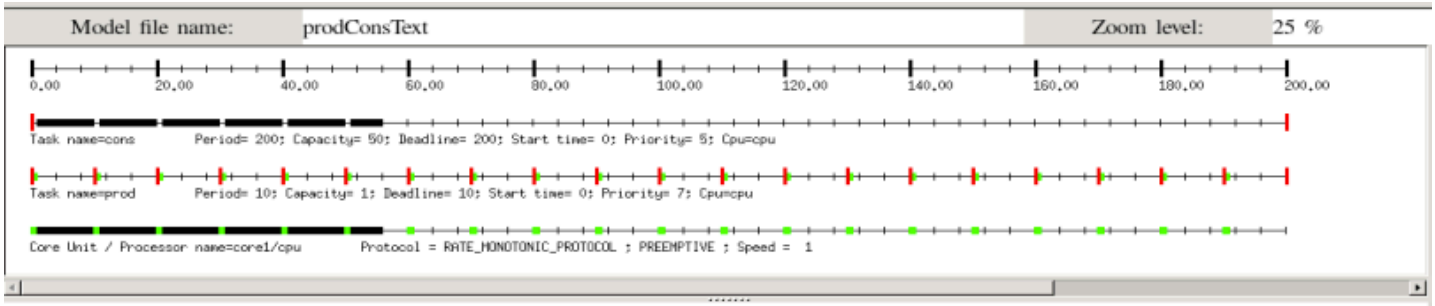
2. How to open a scheduling file in **Cheddar?** If it's not in Program Files, install Cheddar 3.2:
http://beru.univ-brest.fr/svn/CHEDDAR/releases/Cheddar-3.2-Windows-bin.zip

Menu *File/Open* (browse the file). For simulate, click on the green arrow icon; for having the response time, click on gearing icon; for cleaning click on yellow brush. After clicking both icons, you have the window on xxx

More information about Cheddar here:
http://beru.univ-brest.fr/cheddar/ - RefWhatis and here:
http://beru.univ-brest.fr/cheddar/contribs/educational/ubo/ETR-2015-2017-2021/tp-gb.html

If the time diagram (or Gantt diagram) does not appear, drag down the icon [.....] just above the results.
If problem, try Menu *Tools/Scheduling/Customize Scheduling Simulation*. When the window pops out, choose "Graph".

3. Why the time values of the Cheddar models are multiplied by 10?
Cheddar allows only integer numbers for the instants of time, so a capacity of 0.1 cannot be represented.



Scheduling simulation, Processor cpu :

- Number of context switches : 12
- Number of preemptions : 5

- Task response time computed from simulation :
  cons => 56/worst
  prod => 1/worst
- No deadline missed in the computed scheduling : the task set is schedulable if you computed the scheduling on the feasibility interval.

Scheduling feasibility, Processor cpu :

1) Feasibility test based on the processor utilization factor :

- The feasibility interval is 200.0, see Leung and Merill (1980) from [21].
- 130 units of time are unused in the feasibility interval.
- Number of cores hosted by this processor : 1.
- Processor utilization factor with deadline is 0.35000 (see [1], page 6).
- Processor utilization factor with period is 0.35000 (see [1], page 6).
- In the preemptive case, with RM, the task set is schedulable because the processor utilization factor 0.35000 is equal or less than 1.00000 (see [19],

Fig. 11 Cheddar analysis