Modelling and Analysis of Real-Time Systems: Lab Assignments

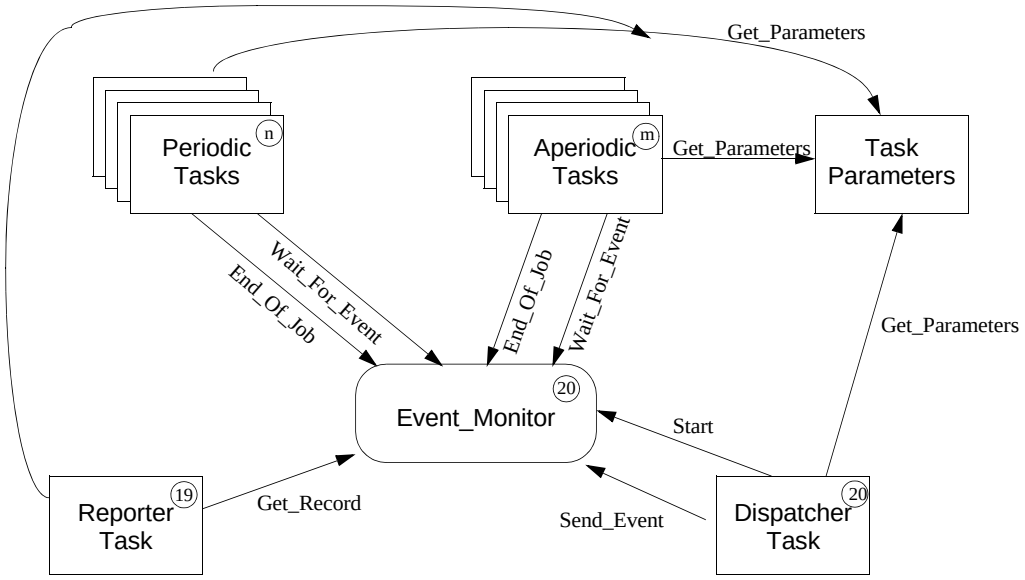
2. Periodic tasks

- Assignment 2.1.a: Periodic Tasks in Ada
- Assignment 2.1.b: Periodic Tasks in C/POSIX

Assignment 2.1.a Periodic Tasks in Ada

1. Develop an application with four periodic tasks, with utilization less than U(4), using a dispatcher task
   - create a main program with the four tasks
   - define the parameters in `task_params.adb`
2. Assign RMA priorities and observe the response times
3. Assign different priorities and check if the deadlines are met or not
4. With the RMA priorities, increase the utilization until the dead-lines are not met
5. Write a report with:
   - source code of the tasks and main program
   - observations on response times under the different cases

Software Architecture



Software Architecture

Three tasks:

- ***User tasks***: periodics (and later, aperiodics)
- ***Dispatcher***: Generates the events that cause the execution of each user task job
- ***Reporter***: generates a report with the results of execution

One protected object

- ***Event monitor***: Stores the events and task job information

Task Parameters

```
with System;
package Task_Params is

   type Task_Count is range 0..10;
   subtype Task_ID is Task_Count range 1..10;

   type Task_Parameters is record
     Period, Exec_Time, Deadline : Duration;
     ... -- other parameters
   end record;

   function Get_Parameters(Of_Task : Task_Id)
     return Task_Parameters;

   function Number_Of_Tasks return Task_Count;

end Task_Params;
```


Dispatcher Task

```
task Dispatcher is
   pragma Priority (20);
end Dispatcher;
```

Dispatcher Task (cont'd)

```
task body Dispatcher is
  Activation_Queue  -- store task ids
                    -- and activation times
  Initial_Time : Time:=Clock;
begin
  for The_Task in All_Tasks loop
    Activation_Queue.Enqueue(The_Task,Initial_Phase);
  end loop;
  Monitor.Start(Initial_Time);
  loop
    Activation_Queue.Dequeue(The_Task,Next_Activation);
    delay until (Initial_Time+Next_Activation);
    Monitor.Send_Event(The_Task);
    Activation_Queue.Enqueue(The_Task,
        Next_Activation+Period(The_Task)); -- for periodics
  end loop;
end Dispatcher;
```


Execution Time Load

```
package Execution_Load is

   procedure Eat (For_Seconds : Duration);

end Execution_Load;
```

Event Monitor

```
with Ada.Calendar, Task_Params;
package Event_Monitor is
   ...
   protected Monitor is
     procedure Start (Initial_Time : in Time);
     procedure Send_Event (The_Task : in Task_ID);
    entry Wait_For_Event (Task_ID);--family of entries
     procedure End_Of_Job (The_Task : in Task_ID);
     procedure Put_Message (Msg  : in String);
     ...
     pragma Priority (20);
   private
     ...
   end Monitor;
end Event_Monitor;
```


Structure of a Periodic Task

```
task type Periodic
   (ID : Task_Id;
    Prio : Priority)
is
   pragma Priority(Prio); -- must be less than 19
end Periodic;

task body Periodic is
begin
   loop
      Monitor.Wait_For_Event(ID);
      Execution_Load.Eat(Exec_Time);
      Monitor.End_Of_Job(ID);
   end loop;
end Periodic;
```

Assignment 2.1.b: Periodic Tasks in C/POSIX

1. Develop an application with four periodic threads, with utilization less than U(4)
2. Assign RMA priorities and observe the response times
3. Assign different priorities and check if the deadlines are met or not
4. With the RMA priorities, increase the utilization until the deadlines are not met
5. Write a report with
   - observations on response times under the different cases

Execution Time Load

File `eat.h`:

```
void eat(const struct timespec *cpu_time);
```

POSIX struct timespec

```
struct timespec {
   time_t tv_sec; // seconds
   long tv_nsec; // nanoseconds
}
```

- measures absolute time since the Epoch,
- or a relative interval

Main Program

The main program does:

- set its own priority to a very high value
  - so that the other threads start at the same time, when the main goes to sleep
- create a threads attributes object and set:
  - inheritsched attribute to `PTHREAD_EXPLICIT_SCHED`
  - detach state to `PTHREAD_CREATE_DETACHED`
  - scheduling policy to `SCHED_FIFO`
- create four periodics threads; for each:
  - change the priority
  - create the thread
- sleep for a very long time

Structure of a Periodic Thread

```
periodic_thread (id, period, exec_time) {
   activation_time = read the clock
   while (1) {
      report ("START" + id + current time)
      eat (exec_time)
      response_time=clock_gettime-activation_time
      report ("END" + id + response_time)
      activation_time=activation_time+period
      sleep until (activation_time)
   }
}
```

Modelling and Analysis of Real-Time Systems: Lab Assignments

3. Extensions to the periodic model

- **Assignment 3.1: Simulating an interrupt**

Assignment 3.1: Simulating an interrupt

Modify the implementation of periodic tasks to simulate an interrupt service routine

- **By changing a medium priority task to the highest priority level**

Compare the response times with those obtained when rate monotonic priorities were used

- **what conclusions can you derive?**

Now try simulating the following solution

- **leave part of the work at interrupt priority**
- **move another part of the work to a new task with rate monotonic priority**
- **compare and discuss the response times obtained**

Modelling and Analysis of Real-Time Systems: Lab Assignments

4. Modelling real-time systems

- **Assignment 4.1: Model and analyze the system developed in Assignment 2.1 or 2.2**
- **Assignment 4.2: Model and analyze the system developed in Assignment 3.1**

Assignment 4.1: Model and analyze the system of Assignment 2.1

Install the MAST toolset and graphical editor

Create a model to represent the system developed in Assignment 2.1 or 2.2

Analyze the response times and check them against the previous results

Write a short report with the results of the analysis

Assignment 4.2: Model and analyze the system of Assignment 3.1

Create a model to represent the system developed in Assignment 3.1

Analyze the response times and check them against the previous results

Write a short report with the results of the analysis

UNIVERSIDAD DE CANTABRIA

Modelling and Analysis of Real-Time Systems: Lab Assignments

5. Synchronization

- **Assignment 5.1.a: Implement task synchronization in the periodic task simulator in Ada**
- **Assignment 5.1.b: Implement task synchronization in the periodic task simulator in C/POSIX**
- **Assignment 5.2: Model and analyze periodic tasks with shared objects**

Assignment 5.1.a: Synchronization

In the task simulation application, create two protected objects to control access to two shared objects

- modify three of the tasks to use these protected objects (one of them using both)
- leave a higher priority task that does not use resources

Do not assign priority ceilings

- the default priority ceiling in Ada is the highest priority
- play with the task phases to achieve the highest blocking times

Assign the appropriate priority ceilings and check the response times

Write a report containing the code of the tasks, protected objects, and main program. Also add a short summary of the results.

Shared Resource Parameters

```
with System;
package Resource_Params is

   type Resource_Count is range 0..2;
   subtype Resource_Id is Resource_Count
       range 1..Resource_Count'Last;

   type Section_Id is range 1..2;

   type Exec_Times is array (Section_Id) of Duration;

   type Resource_Parameters is record
      Prio : System.Priority;
      Exec : Exec_Times;
   end record;
```


Shared Resource Parameters (cont'd)

```
   function Get_Parameters
     (Of_Resource : Resource_Id)
     return Resource_Parameters;

   function Number_Of_Resources
     return Resource_Count;

end Resource_Params;
```

Specification of the Protected Object

```
protected type Shared_Resource(Ceiling :Priority) is

   procedure Critical_Sect_1 (Length : in Duration);
   procedure Critical_Sect_2 (Length : in Duration);

   pragma Priority (Ceiling);
  // Comment out the line above to use default ceiling

end Shared_Resource;
```


Body of the Protected Object

```
protected body Shared_Resource is

   procedure Critical_Sect_1
      (Length : in Duration) is
   begin
      Execution_Load.Eat(Length);
   end Critical_Sect_1

   procedure Critical_Sect_2
      (Length : in Duration) is
   begin
      Execution_Load.Eat(Length);
   end Critical_Sect_2

end Shared_Resource;
```

Periodic Task that Uses Shared Objects

```
task body Periodic_With_Synch is
begin
   Exec_Time:=Exec_Time-Length of critical section
   loop
      Monitor.Wait_For_Event(ID);
      Execution_Load.Eat(Exec_Time/2);

      Resource_Critical_Sect_1(Length);

      Execution_Load.Eat(Exec_Time/2);
      Monitor.End_Of_Job(ID);
   end loop;
end Periodic;
```


Assignment 5.1.b: Implement task synchronization in C/POSIX

A task simulation application that uses one mutex to control access to a shared object

- the shared object is coded in a monitor with operations that use the mutex
- the mutex uses the immediate priority ceiling protocol
  - the ceiling is adjustable at initialization time
- two tasks use operations of the shared object

Assignment 5.1.b (cont'd)

Adapt this application to experiment with priority inversion

- create a medium priority task that does not use the shared object
- change the mutex protocol to PTHREAD_PRIO_NONE
- play with the task phases to observe the unbounded priority inversion effect

Then

- use the PTHREAD_PRIO_PROTECT protocol for the mutex
- assign the appropriate priority ceilings
- check the response times

Write a report with the developed code and a short summary of the results

Assignment 5.2: Model and analyze periodic tasks with shared objects

Build a MAST model of the simulation application with shared resources, and analyze it

Compare the results with those observed in practice

Write a report with the results obtained, including the response times and blocking times

Modelling and Analysis of Real-Time Systems: Lab Assignments

6. Aperiodic events

- Assignment 6.1.a. Aperiodic tasks in Ada
- Assignment 6.1.b. Aperiodic tasks in C/POSIX

Assignment 6.1.a: Aperiodic tasks in Ada

- Create one aperiodic task
- Observe the unpredictability introduced by this task
- Use the polling server approach, and check the predictability of the response times
- Use the emulated sporadic server approach
- Build a MAST model of the system and obtain the response times

Specification of aperiodic tasks

```
with System;
with Resource_Params;

package Task_Params is

   type Task_Count is range 0..10;
   subtype Task_Id is Task_Count range 1..Task_Count'Last;

   type Event_Kinds is (Periodic, Aperiodic);

   type Task_Kinds is (Regular, With_Sync, Polling, Sporadic);
```


Specification of aperiodic tasks (cont'd)

```
   type Task_Parameters is record
      Event_Kind : Event_Kinds;
      Task_Kind : Task_Kinds;
      Prio : System.Priority;
      Period, Exec_Time, Deadline, Phase,
        Avg_Interarrival, Initial_Capacity : Duration;
      Res_Id : Resource_Params.Resource_ID;
      Sect_Id  : Resource_Params.Section_Id;
   end record;

   function Get_Parameters (Of_Task : Task_Id)
       return Task_Parameters;

   function Number_Of_Tasks return Task_Count;

end Task_Params;
```

Simple Aperiodic Task

```
task type Aperiodic
   (ID   : task_id;
    Prio : Priority)
is
    pragma Priority(Prio);
end Aperiodic;

task body Aperiodic is
begin
   loop
     Monitor.Wait_For_Event(ID);
     Execution_Load.Eat(Exec_Time);
     Monitor.End_Of_Job(ID);
   end loop;
end Aperiodic;
```


Aperiodic Task with Polling

```
task body Aperiodic is
   Next_Activation : Time:=Clock;
begin
   loop
     select
        Monitor.Wait_For_Event(ID);
        Execution_Load.Eat(Exec_Time);
        Monitor.End_Of_Job(ID);
     else
        null;
     end select;
     Next_Activation:=Next_Activation+Period;
     delay until Next_Activation;
   end loop;
end Aperiodic;
```

Aperiodic Task with Emulated Sporadic Server

```
task body Aperiodic is
   Capacity : Duration:=Initial_Capacity;
begin
   loop
      Monitor.Wait_For_Event(ID);
      Last_Activation:=Clock;
      Execution_Load.Eat(Exec_Time);
      Monitor.End_Of_Job(ID);
      Capacity:=Capacity-Exec_Time;
      if Capacity<Exec_Time then
         delay until Last_Activation+Period;
         Capacity:=Initial_Capacity;
      end if;
   end loop;
end Aperiodic;
```


Assignment 6.1.b: Aperiodic tasks in C/POSIX

- Use the proposed application with one aperiodic thread and two periodic threads
  - the events arrive with a random interarrival time
- Observe the unpredictability introduced by this thread
- Use the polling server approach, and check the predictability of the response times
- Use the emulated sporadic server approach
- Build a MAST model of the system and obtain the response times

Structure of an Aperiodic Thread

```
aperiodic_thread (id, signal, avg_interarrival,
                  exec_time)
{
   next_event= initial_time+phase;
   while (1) {
      // simulate the wait for an aperiodic event
      clock_nanosleep(ABS_TIME,next_event);
      // process the event
      report ("START" + id + current period)
      eat (exec_time)
      response_time=clock_gettime-next_event
      report ("END" + id + response_time)
      // simulate arrival of next event
      next_event= next_event+
          random(0..2*avg_interarrival);
   }
}
```


Structure of an Aperiodic Thread with Polling

```
aperiodic_thread (id, signal, avg_interarrival, polling_period,
                  exec_time)
{
   next_event= initial_time+phase;
   next_activation= next_event
   next_event= next_event+random(0..2*avg_interarrival);
   while (1) {
      // simulate the arrival of an aperiodic event
      now=clock_gettime
      if (now>=next_event) {
          // event has arrived
          report ("START" + id + current period)
          eat (exec_time)
          response_time=clock_gettime-next_event
          report ("END" + id + response_time)
          next_event=next_event+random(0..2*avg_interarrival);
      }
```

Structure of an Aperiodic Thread with Polling

```
        // wait for next period
        next_activation=next_activation+polling_period
        clock_nanosleep(ABS_TIME,next_activation);
    }
}
```


Structure of an Aperiodic Thread with Emulated Sporadic Server

```
aperiodic_thread (id, signal, avg_interarrival,
                  initial_capacity, repl_period, exec_time)
{
    next_event= initial_time+phase;
    capacity=initial_capacity
    while (1) {
        // simulate the wait for an aperiodic event
        next_event= next_event+
            random(0..2*avg_interarrival);
        clock_nanosleep(ABS_TIME,next_event);
        last_activation=clock_gettime;
        // process the event
        report ("START" + id + current period)
        eat (exec_time)
        response_time=clock_gettime-next_event
        report ("END" + id + response_time)
        capacity=capacity-exec_time
```

Structure of an Aperiodic Thread with Emulated Sporadic Server

```
        // check available capacity
        if (capacity<exec_time) {
           clock_nanosleep(ABS_TIME,
               last_Activation+repl_period)
           capacity=initial_capacity
        }
    }
}
```

7. Dynamic priority scheduling

- **Assignment 7. Use MAST to model and analyze the system described in Exercise 7**

Modelling and Analysis of Real-Time Systems: Lab Assignments

8. Advanced topics

- **Assignment 8. Model and analyze a system with periodic tasks with release jitter and post-period deadlines.**

Assignment 8. Release Jitter and Post-Period Deadlines

Suppose the following system:

	C	T	D	J
Task τ_1:	20	100	100	10
Task τ_2:	25	150	150	10
Task τ_3:	100	350	700	10

We now want to add a new periodic transaction that does the following actions

- RS: read a sensor
- PR: process the information from the sensor
- OA: output the result to an actuator

- **The transaction is periodic, with a period of 50 ms and 1ms release jitter**

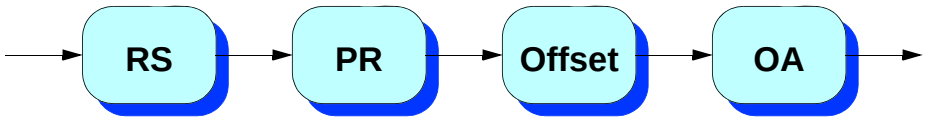
Assignment 8 (cont'd)

The requirements of the new transaction are

	C	D	Max Output Jitter
RS	1	-	3
PR	5	-	
OA	1	20	3

Assignment 8 (cont'd)

To accomplish these requirements we design the transaction as follows



The offset is set to the worst-case response time of activity PR.

RS and OA are given the highest priority

PR is given the next highest priority

Assignment 8 (cont'd)

Steps:

- create a MAST model of this system
- analyze it to obtain the correct value of the offset
- make a final analysis to check whether the system is schedulable or not
- write a short report with the results obtained

UNIVERSIDAD DE CANTABRIA

Appendix 1: Using the Gnat Compiler

To compile with gnat-2007:

```
. /usr/local/gnat-2007/gnatvars.sh
```

To compile a module called `name.adb`:

```
gcc -c -gnatv name.adb
```

To compile everything and link the program called `name`

```
gnatmake -gnatv name.adb
```

The same, using the MaRTE OS run-time system

```
gnatmake -gnatv --RTS=marte name.adb
```


Appendix 1: Using the Gnat Compiler

Main compiler options

- `-gnatv`: shows the lines with errors
- `-g`: needed to use the debugger
- `-gnato`: to check the results of arithmetic operations
- example: `gnatmake -gnatv -gnato -g name.adb`

Appendix 2: Using MaRTE OS in Linux

Set the environment to compile with *MaRTE OS* under the *linux* architecture:

```
. /usr/local/marte-1.8/martevars.sh
```

Compile a C module called `name.c`:

```
mgcc -c name.c
```

Compile a C program called `prog.c` and link it with a module called `name.c`:

```
mgcc prog.c name.o
```

Execute the program:

```
./mprogram
```


Appendix 3: Using MAST

To have access to MAST:

```
. /usr/local/mast-1.4.0.1/mastvars.sh
```