POLITECNICO DI TORINO

OSES LAB #3 – Scheduling, Priority Ceiling, Deadlock

Department of CONTROL AND COMPUTER ENGINEERING (DAUIN)
Master's degree in Mechatronic Engineering
2023/2024
Operating systems for embedded systems– Prof. Violante
Laboratory 03
Wednesday, 30 November 2023

Student:
s320296 DI MARTINO GIULIO

Exercise #1

Define three periodic tasks with implicit deadline (deadline equal to period) and the following parameters:

Task	*Period*	*Execution time*
A	1200ms	200ms
B	2000ms	600ms
C	3000ms	300ms

All tasks must be ready for execution at t=0. Use the function `do_things(ms)` to simulate execution time. It traps the current task in a busy loop for `ms` milliseconds.

C-CODE:

```
#include "tpl_os.h"
#include "Arduino.h"


void setup()
{

    Serial.begin(115200);

}


  void do_things(int ms)
{
 unsigned long mul = ms * 200UL;
 unsigned long i;
 for(i=0; i<mul; i++) millis();
 }





TASK(TaskA)
{
    static int a = 0;
    Serial.print("\n");
    Serial.print("[taskA_");
    Serial.print(a);
    Serial.print("]");
    Serial.print("\t");
    Serial.print(millis());
    Serial.print("-->");
    do_things(200);
    Serial.println(millis());
    Serial.print("\n");
    a++;



}


TASK(TaskB)
{
    static int b = 0;
    Serial.print("[taskB_");
    Serial.print(b);
    Serial.print("]");
    Serial.print("\t");
    Serial.print(millis());
    Serial.print("-->");
    do_things(600);
    Serial.println(millis());
    Serial.print("\n");
    b++;

}


TASK(TaskC)
{
    static int c = 0;
    Serial.print("[taskC_");
    Serial.print(c);
    Serial.print("]");
    Serial.print("\t");
    Serial.print(millis());
    Serial.print("-->");
    do_things(300);
    Serial.println(millis());
    Serial.print("\n");
    c++;


}
```

This C-code is divide into three task and a function called “void do_things()”
The function is useful to simulate that the tasks do something in a certain execution time so I send a int to this function with the value of each task’s time.

```
  void do_things(int ms)
{
 unsigned long mul = ms * 200UL;
 unsigned long i;
 for(i=0; i<mul; i++) millis();
 }
```

Furthermore, in the tasks I set some Serial.print to see if they work properly.

OIL FILE:

```

OIL_VERSION = "2.5" : "test" ;

CPU test {
  OS config {
    STATUS = STANDARD;
    BUILD = TRUE {
      TRAMPOLINE_BASE_PATH = "../../../..";
      APP_NAME = "lab03.1";
      APP_SRC = "lab03_1.cpp";
      CPPCOMPILER = "avr-g++";
      COMPILER = "avr-gcc";
      LINKER = "avr-gcc";
      ASSEMBLER = "avr-gcc";
      COPIER = "avr-objcopy";
      SYSTEM = PYTHON;
      LIBRARY = serial;

    };

  };

  APPMODE stdAppmode {};

 ALARM a1200sec {
    COUNTER = SystemCounter;
    ACTION = ACTIVATETASK { TASK = TaskA; };
    AUTOSTART = TRUE { APPMODE = stdAppmode; ALARMTIME = 1172; CYCLETIME = 1172; APPMODE =
stdAppmode;};
  };

ALARM a2000msec {
    COUNTER = SystemCounter;
    ACTION = ACTIVATETASK { TASK = TaskB; };
    AUTOSTART = TRUE { APPMODE = stdAppmode; ALARMTIME = 1953; CYCLETIME = 1953; APPMODE =
stdAppmode;};
  };

 ALARM a3000msec{
    COUNTER = SystemCounter;
    ACTION = ACTIVATETASK{TASK = TaskC;};
    AUTOSTART = TRUE{APPMODE = stdAppmode; ALARMTIME = 2930;CYCLETIME = 2930; APPMODE =
stdAppmode;};
  };

  TASK TaskA {
    PRIORITY = 3;
    AUTOSTART = TRUE {APPMODE = stdAppmode; };
    ACTIVATION = 1;
    SCHEDULE = FULL;
  };

 TASK TaskC {
    PRIORITY = 1;
    AUTOSTART = TRUE { APPMODE = stdAppmode; };
    ACTIVATION = 1;
    SCHEDULE = FULL;
  };


TASK TaskB {
    PRIORITY = 2;
    AUTOSTART = TRUE { APPMODE = stdAppmode; };
    ACTIVATION = 1;
    SCHEDULE = FULL;
  };

};
```

In the oil file I set three alarms:

```
ALARM a1200sec {
    COUNTER = SystemCounter;
    ACTION = ACTIVATETASK { TASK = TaskA; };
    AUTOSTART = TRUE { APPMODE = stdAppmode; ALARMTIME = 1172; CYCLETIME = 1172; APPMODE =
stdAppmode;};
  };

ALARM a2000msec {
    COUNTER = SystemCounter;
    ACTION = ACTIVATETASK { TASK = TaskB; };
    AUTOSTART = TRUE { APPMODE = stdAppmode; ALARMTIME = 1953; CYCLETIME = 1953; APPMODE =
stdAppmode;};
  };

 ALARM a3000msec{
    COUNTER = SystemCounter;
    ACTION = ACTIVATETASK{TASK = TaskC;};
    AUTOSTART = TRUE{APPMODE = stdAppmode; ALARMTIME = 2930;CYCLETIME = 2930; APPMODE =
stdAppmode;};
  };
```

Each alarm activates the task at t=0 with a periodic time equal to the period of the task; the time in the alarm instruction is in this configuration because a 16Mhz Arduino board performs a tick every 1024 µs, so:

$$1024: 1000 = time_{desired}: X$$

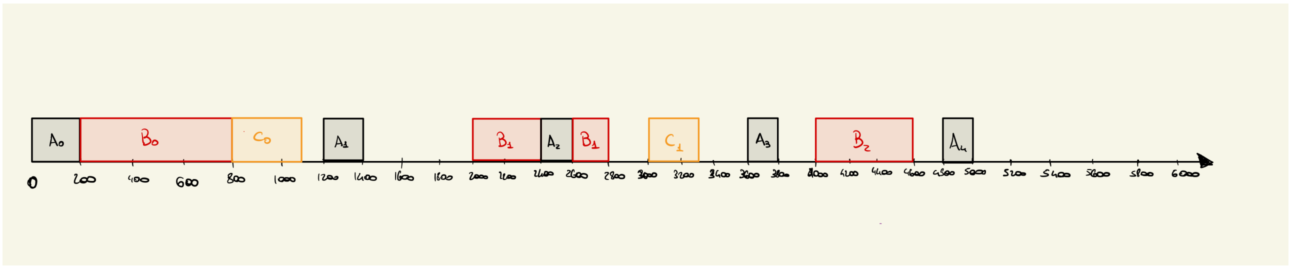
AUTOSTART =TRUE so the Task is placed in the ready list upon started the Operating System.

```
TASK TaskA {
    PRIORITY = 3;
    AUTOSTART = TRUE {APPMODE = stdAppmode; };
    ACTIVATION = 1;
```

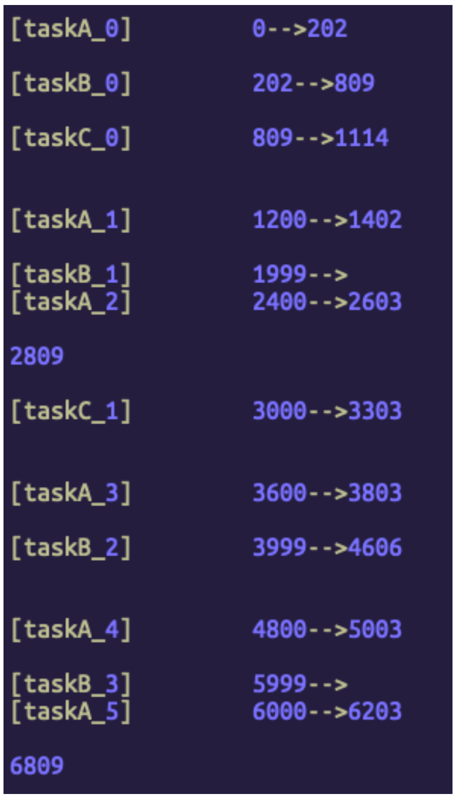
```
     SCHEDULE = FULL;
   };

  TASK TaskC {
     PRIORITY = 1;
     AUTOSTART = TRUE { APPMODE = stdAppmode; };
     ACTIVATION = 1;
     SCHEDULE = FULL;
   };


 TASK TaskB {
     PRIORITY = 2;
     AUTOSTART = TRUE { APPMODE = stdAppmode; };
     ACTIVATION = 1;
     SCHEDULE = FULL;
   };

 };
```

I set the priority in this way to follow the RM method, so the task with the minor period is set to have the high priority and so on.

Theoretically we shall obtain this result:



Simulate with SimulIDE I obtain almost the same result as I expected in the theoretic once:

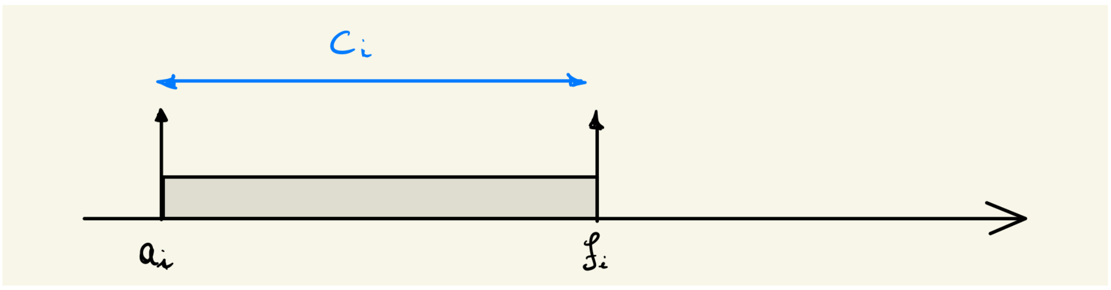


```
[taskA_0]          0-->202

[taskB_0]          202-->809

[taskC_0]          809-->1114


[taskA_1]          1200-->1402

[taskB_1]          1999-->
[taskA_2]          2400-->2603

2809

[taskC_1]          3000-->3303


[taskA_3]          3600-->3803

[taskB_2]          3999-->4606


[taskA_4]          4800-->5003

[taskB_3]          5999-->
[taskA_5]          6000-->6203

6809
```

The *worst-case response time* of task C:

- Theoretically: is the different between the initial time a_i and the finish time f_i So, in our case is 300 ms.

- Reality: I measure that in simulIDE, I print the initial value and the final value for enough time, and I take the worst case is almost 303.



Exercise #2

Update the code of exercise #1 so that A and C share a resource in the following way:

- On each activation, A requests the resource, keeps it for 300ms, and releases it.
- On each activation, C requests the resource after executing for 200ms, keeps it for 300ms, and releases it.

C-CODE:

```
#include <time.h>
#include <stdio.h>
#include <math.h>
#include "tpl_os.h"
#include "Arduino.h"
#include "tpl_com.h"

DeclareAlarm(a1200msec);
DeclareAlarm(a2000msec);
DeclareAlarm(a3000msec);

DeclareResource(ResA);


void do_things(int ms=1){
    unsigned long mul = ms * 200UL;
    unsigned long i;
    for(i=0; i<mul; i++)
        millis();
}

void setup()
{
       Serial.begin(115200);
}

TASK(TaskA){
    static int a=0;
    Serial.print("\n");
    Serial.print("[taskA_");
    Serial.print(a);
    Serial.print("]");
    Serial.print("\t");
    Serial.print(millis());
    Serial.print("-->");
    GetResource(ResA); //definisci su oil
    do_things(300);
    Serial.println(millis());
    Serial.print("\n");
    a++;
    ReleaseResource(ResA);

}

TASK(TaskB)
{
    static int b = 0;
    Serial.print("[taskB_");
    Serial.print(b);
    Serial.print("]");
    Serial.print("\t");
    Serial.print(millis());
    Serial.print("-->");
    do_things(600);
    Serial.println(millis());
    Serial.print("\n");
    b++;
}
```

```
TASK(TaskC){
    static int k=0;
    Serial.print("[taskC_");
    Serial.print(k);
    Serial.print("]");
    Serial.print("\t");
    Serial.print(millis());
    Serial.print("-->");
    do_things(200);
    GetResource(ResA); //definisci su oil
    do_things(300);
    Serial.println(millis());
    Serial.print("\n");
    ReleaseResource(ResA);
    k++;

    }
```

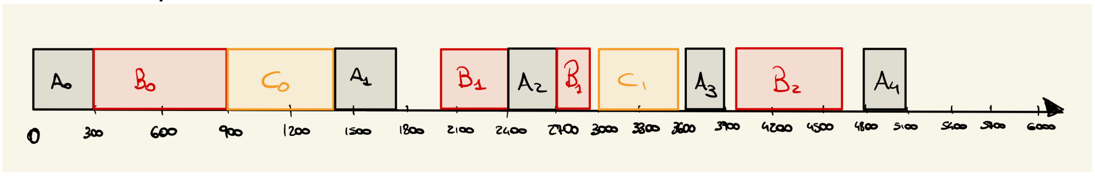
This code is the same as before we set a resource called **Res** this resource is the same in the task A and task C; in this way when these tasks enter in the critical section, they don't have preemption, so we don't lost any information.

```
OIL_VERSION = "2.5" : "test" ;

CPU test {
  OS config {
    STATUS = STANDARD;
    BUILD = TRUE {
      TRAMPOLINE_BASE_PATH = "../../../..";
      APP_NAME = "lab3_2";
      APP_SRC = "lab03_2.cpp";
      CPPCOMPILER = "avr-g++";
      COMPILER = "avr-gcc";
      LINKER = "avr-gcc";
      ASSEMBLER = "avr-gcc";
      COPIER = "avr-objcopy";
      SYSTEM = PYTHON;

      LIBRARY = serial;

    };
    SYSTEM_CALL = TRUE;
  };

  APPMODE stdAppmode {};

RESOURCE Res{
      RESOURCEPROPERTY = STANDARD;
};


ALARM a1200sec {
    COUNTER = SystemCounter;
    ACTION = ACTIVATETASK { TASK = TaskA; };
    AUTOSTART = TRUE { APPMODE = stdAppmode; ALARMTIME =1172; CYCLETIME =1172; };
  };

ALARM a2000msec {
    COUNTER = SystemCounter;
    ACTION = ACTIVATETASK { TASK = TaskB; };
    AUTOSTART = TRUE { APPMODE = stdAppmode; ALARMTIME = 1953; CYCLETIME =1953; };
  };

 ALARM a3000msec{
    COUNTER = SystemCounter;
    ACTION = ACTIVATETASK{TASK = TaskC;};
    AUTOSTART = TRUE{APPMODE = stdAppmode;ALARMTIME =  2930;CYCLETIME = 2930; };
  };

  TASK TaskA {
    PRIORITY = 3;
    AUTOSTART = TRUE {APPMODE = stdAppmode; };
    ACTIVATION = 1;
    SCHEDULE = FULL;
    RESOURCE= Res;


  };


 TASK TaskB {
    PRIORITY = 2;
    AUTOSTART = TRUE { APPMODE = stdAppmode; };
    ACTIVATION = 1;
    SCHEDULE = FULL;
  };



 TASK TaskC {
    PRIORITY = 1;
    AUTOSTART = TRUE { APPMODE = stdAppmode; };
    ACTIVATION = 1;
    SCHEDULE = FULL;
    RESOURCE= Res;

  };

};
```

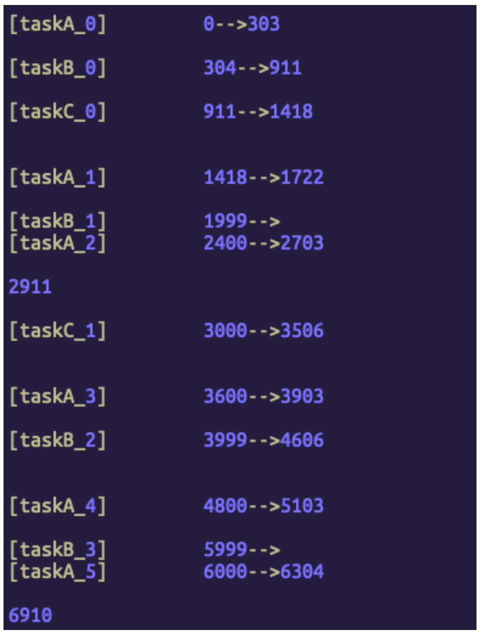
```
RESOURCE Res{
      RESOURCEPROPERTY = STANDARD;
};
```

RESOURCEPROPERTY = STANDARD means that the resource is associated with 2 system calls (*GetResource()* and *ReleaseResource()*).

Theoretically we shall obtain this result:



Simulate with SimulIDE I obtain almost the same result as I expected in the theoretic once:



```
[taskA_0]          0-->303

[taskB_0]          304-->911

[taskC_0]          911-->1418


[taskA_1]          1418-->1722

[taskB_1]          1999-->
[taskA_2]          2400-->2703

2911

[taskC_1]          3000-->3506


[taskA_3]          3600-->3903

[taskB_2]          3999-->4606


[taskA_4]          4800-->5103

[taskB_3]          5999-->
[taskA_5]          6000-->6304

6910
```

We can see that when the task C is activated and task A want to start, they shall attend the termination of the task C so we don't have preemption between the Task A and Task C (*priority ceiling)*, we can see preemption between Task B and Task A because these two tasks don't share resource.

For example, at 1200 ms task A_1 doesn't preempt the task C_0 because C_0 is keeping the resource. The priority ceiling comes into effect.

At 2400 ms the task B_1 is preempted by the Task A_2 due the fact that we don't use resource, so they are affected by the different value of priority.

Exercise #3

Mr. C. Lever believes he can use a message to implement critical regions in tasks A and C, instead of using a resource like in exercise #2, in this way:

- At t=0 an initialization task runs once and sends one message to a message object.
- When task A or C want to enter their critical section, they repeatedly check the

  message object until they successfully receive the message.

- They send back the message at the end of their critical section.
- Message type and content are irrelevant.

C – CODE

```
#include <time.h>
#include <stdio.h>
#include <math.h>
#include "tpl_os.h"
#include "Arduino.h"
#include "tpl_com.h"


DeclareAlarm(a1200msec);
DeclareAlarm(a2000msec);
DeclareAlarm(a3000msec);

DeclareMessage(Message_send_task);
DeclareMessage(Message_rec_task);



void do_things(int ms=1){
    unsigned long mul = ms * 200UL;
    unsigned long i;
    for(i=0; i<mul; i++)
        millis();
}

void setup()
{
       Serial.begin(115200); //115200 bps, 8N1
}

TASK(TaskA){
    static int a = 0;
    static int recive_a;
    Serial.print("\n");
    Serial.print("[taskA_");
    Serial.print(a);
    Serial.print("]");
    Serial.print("\t");
    Serial.print(millis());
    Serial.print("-->");
    while  (ReceiveMessage(Message_rec_task, &recive_a) != E_OK) {
        Serial.print("[TASKA] error\n");
        }
    do_things(300);

    SendMessage(Message_send_task,&recive_a);


    Serial.println(millis());
    Serial.print("\n");
    a++;


    }
```

```
TASK(TaskB){
    static int b=0;
    Serial.print("[taskB_");
    Serial.print(b);
    Serial.print("]");
    Serial.print("\t");
    Serial.print(millis());
    Serial.print("-->");
    do_things(600);
    Serial.println(millis());
    Serial.print("\n");
    b++;

}

TASK(TaskC){
    static int recive_c;
    static int c = 0;
    Serial.print("[taskC_");
    Serial.print(c);
    Serial.print("]");
    Serial.print("\t");
    Serial.print(millis());
    Serial.print("\t");
    do_things(200);
    Serial.print("-->");
    while(ReceiveMessage(Message_rec_task, &recive_c) != E_OK) {
        Serial.print("[TASKC] error\n");
    }
    do_things(300);
    SendMessage(Message_send_task,&recive_c);
    Serial.print(" Finish Time : ");
    Serial.println(millis());
    Serial.print("\n");
    c++;

}

TASK(Initialize) {
    static int send=3;
    SendMessage(Message_send_task,&send);

    Serial.print("Initialize\t");
    Serial.print(send);

}
```

I use another task to initialize the message, this task is set whit the high priority as we can see in the oil file:

```
TASK(Initialize) {
    static int send=3;
    SendMessage(Message_send_task,&send);

    Serial.print("Initialize\t");
    Serial.print(send);

}
```

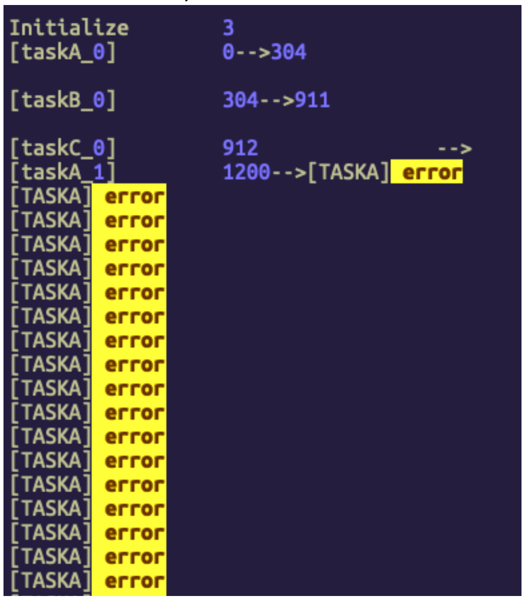
OIL FILE:

```
OIL_VERSION = "2.5" : "test" ;

CPU test {
  OS config {
    STATUS = STANDARD;
    BUILD = TRUE {
      TRAMPOLINE_BASE_PATH = "../../../..";
      APP_NAME = "lab3_3";
      APP_SRC = "lab03_3.cpp";
      CPPCOMPILER = "avr-g++";
      COMPILER = "avr-gcc";
      LINKER = "avr-gcc";
      ASSEMBLER = "avr-gcc";
      COPIER = "avr-objcopy";
      SYSTEM = PYTHON;

      LIBRARY = serial;

    };
    SYSTEM_CALL = TRUE;
  };

  APPMODE stdAppmode {};


MESSAGE Message_send_task {
  MESSAGEPROPERTY = SEND_STATIC_INTERNAL{
     CDATATYPE = "int";
  };
};





MESSAGE Message_rec_task{
    MESSAGEPROPERTY = RECEIVE_QUEUED_INTERNAL{
      SENDINGMESSAGE = Message_send_task;
      QUEUESIZE=1;
    };
  };





ALARM a1200sec {
    COUNTER = SystemCounter;
    ACTION = ACTIVATETASK { TASK = TaskA; };
    AUTOSTART = TRUE { APPMODE = stdAppmode; ALARMTIME =1172; CYCLETIME =1172; };
  };

ALARM a2000msec {
    COUNTER = SystemCounter;
    ACTION = ACTIVATETASK { TASK = TaskB; };
    AUTOSTART = TRUE { APPMODE = stdAppmode; ALARMTIME = 1953; CYCLETIME =1953; };
  };

 ALARM a3000msec{
    COUNTER = SystemCounter;
    ACTION = ACTIVATETASK{TASK = TaskC;};
    AUTOSTART = TRUE{APPMODE = stdAppmode;ALARMTIME =  2930;CYCLETIME = 2930; };
  };

  TASK TaskA {
    PRIORITY = 3;
    AUTOSTART = TRUE
    {
      APPMODE = stdAppmode;
    };
    ACTIVATION = 1;
    SCHEDULE = FULL;


  };

 TASK TaskC {
    PRIORITY = 1;
    AUTOSTART = TRUE { APPMODE = stdAppmode; };
    ACTIVATION = 1;
    SCHEDULE = FULL;
  };


TASK TaskB {
    PRIORITY = 2;
    AUTOSTART = TRUE { APPMODE = stdAppmode; };
    ACTIVATION = 1;
    SCHEDULE = FULL;
  };
TASK Initialize{
    PRIORITY = 10;
    AUTOSTART = TRUE{ APPMODE = stdAppmode; };
    ACTIVATION = 1;
    SCHEDULE = FULL;
};

};
```

```
MESSAGE Message_send_task {
  MESSAGEPROPERTY = SEND_STATIC_INTERNAL{
     CDATATYPE = "int";
  };
};





MESSAGE Message_rec_task{
    MESSAGEPROPERTY = RECEIVE_QUEUED_INTERNAL{
      SENDINGMESSAGE = Message_send_task;
      QUEUESIZE=1;
    };
  };
```

We use a message queued with queue one this message work between the task A e C. **Message_send_task** ,the data type is an **int16,** the property is **SEND_STATIC_INTERNAL** so it's possible to exchange data only for the tasks that run in the same processor.

So, this code works as the exercise 2 where instead of using the resource I use the message. When I run the code, I obtain:



```
Initialize        3
[taskA_0]         0-->304

[taskB_0]         304-->911

[taskC_0]         912              -->
[taskA_1]         1200-->[TASKA] error
[TASKA] error
[TASKA] error
[TASKA] error
[TASKA] error
[TASKA] error
[TASKA] error
[TASKA] error
[TASKA] error
[TASKA] error
[TASKA] error
[TASKA] error
[TASKA] error
[TASKA] error
[TASKA] error
[TASKA] error
[TASKA] error
[TASKA] error
```

We can see that all the tasks run at t=0, but when the code check the message, it found an error in the task A because the task C don't resend the message to the task A.
So, if Mr. Lever want to implement critical regions this isn't a good way. He can use a resource like the exercise 2.