UNIVERSITÀ DI BOLOGNA

School of Engineering

Master Degree in Automation Engineering

REAL TIME SYSTEMS FOR AUTOMATION M

ASSIGNMENT 3 (RT): CREATE AND ANALYZE YOUR OWN APPLICATION

Professors:
Daniela Loreti
Paolo Torroni

Students:
Luca Santoro 0001005415
Armando Spennato 0001006172
Riccardo Bassi 0001084538

Academic year 2022/2023

Chapter 1

Task set definition

This assignment consists of implementing and analyzing an application of your own compilation and commenting on its behavior. The analysed application is composed of 3 harmonic periodic tasks which share 2 different, non-preemptable resources R_a and R_b; The considered application of three tasks is scheduled considering just a single processor machine, applying the Priority Inheritance Protocol (**PIP**) to solve the priority inversion problem. In figure 1.1 is shown a graphical representation of the critical sections in Γ_1. In the table are reported the phase, the worst-case computation time and period of each task ($T_i = D_i$).

Γ_1	ϕ_i	T_i	C_i	$t(R_a)$	δ_a	$t(R_b)$	δ_b
τ_1	0	4	1	0	1	-	-
τ_2	0	8	3	0,5	2	0	2,5
τ_3	0	16	5	1	3	0	4,5

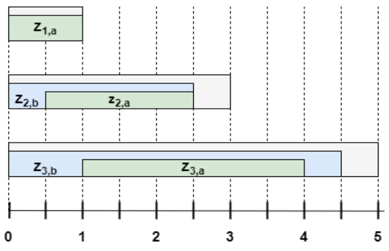


Figure 1.1: Γ_1 Task set definition.

The detail of accesses to shared resources is as follows:

- τ_1 : immediately acquires resource R_a and holds it for 1 units and terminates;
- τ_2 : immediately acquires resource R_b, after 0,5 unit acquires also resource R_a, holds it for 2 units; then releases both resources and after 0.5 unit terminates;
- τ_3 : immediately acquires resource R_b; after 1 unit acquires also resource R_a, holds it for 3 units then releases it, after 0,5 unit releases resource R_b, 0,5 unit later finally terminates.

Chapter 2

Feasibility Analysis

2.1 Blocking Time Computaton

To verify the feasibility of the task set we have to do the extended analysis. As first step we find the Blocking times shown in 2.1.

Task	δ_a	δ_b	B_i
τ_1	1	-	6,5
τ_2	2	2,5	4,5
τ_3	3	4,5	0

Table 2.1: Blocking times.

In particular, since we are dealing with the PIP protocol, we are able to find only a bound, not the exact value for the blocking times. Are just an approximation which try to figure out what is the possible worst-case scenario considering that each task can be blocked by each other lower priority task. By considering the first two columns of table 2.1, we have to select all the possible combinations we can get by choosing 1 element for each column/row: then we sum up the values selected by each combination we have found. The result of our computation is the highest number among all the possible combinations.

2.2 Extended Static Priority Ordering Analysis

At this point we can stat with our analysis:

- **Task 1:**
  $\tau_1 \rightarrow \frac{C_1+B_1}{T_1} = \frac{1+6.5}{4} = 1.875\ (> 1)$ **NO**

- **Task 2:**

  $\tau_2 \rightarrow \frac{C_1}{T_1} + \frac{C_2+B_2}{T_2} = \frac{1}{4} + \frac{3+4.5}{8} = 1.1875 \ (> 1)$ **NO**

- **Task 3:**

  $\tau_3 \rightarrow \frac{C_1}{T_1} + \frac{C_2}{T_2} + \frac{C_3+B_3}{T_3} = \frac{1}{4} + \frac{3}{8} + \frac{5+0}{16} = 0.9375 \ (\leq 1)$ **OK**

2.3 Extended Response Time Analysis

- **Task 1**

  $\tau_1 \rightarrow R_1^0 = C_1 + B_1 = 1 + 6.5 = 7.5$

  $R_1^0 = 7.5 > T_1 = 4$ **NO**

- **Task 2**

  $\tau_2 \rightarrow R_2^0 = C_1 + C_2 + B_2 = 1 + 3 + 4.5 = 8.5$

  $R_2^0 = 8.5 > T_2 = 8$ **NO**

- **Task 3**

  $\tau_3 \rightarrow R_3^0 = C_1 + C_2 + C_3 + B_3 = 1 + 3 + 5 + 0 = 9$

  $R_3^0 = 9 < T_3 = 16$ **OK**

  $R_3^1 = \left\lceil \frac{R_3^0}{T_1} \right\rceil C_1 + \left\lceil \frac{R_3^0}{T_2} \right\rceil C_2 + C_3 + B_3 = 3 + 6 + 5 + 0 = 14$

  $(R_3^1 > R_3^0)$

  $R_3^2 = \left\lceil \frac{R_3^1}{T_1} \right\rceil C_1 + \left\lceil \frac{R_3^1}{T_2} \right\rceil C_2 + C_3 + B_3 = 4 + 6 + 5 + 0 = 15$

  $(R_3^2 > R_3^1)$

  $R_3^3 = \left\lceil \frac{R_3^2}{T_1} \right\rceil C_1 + \left\lceil \frac{R_3^1}{T_2} \right\rceil C_2 + C_3 + B_3 = 4 + 6 + 5 + 0 = 15$

  $(R_3^3 = R_3^2)$ $R_3^3 = 15 < T_3 = 16$ **OK**

2.4 Analysis Result

As we can see, both in the extended static priority ordering analysis and the extended response time analysis we have that the test fails for task 1 and task 2. For this reason we cannot conclude anything about feasibility.

Chapter 3

Scheduling Diagrams

To compare the results given by VxWorkS, using a Gantt chart, we have drawn the schedule by hand, this can be shown in figure 3.1. In the figure are also reported the priority of task τ_2 and τ_3.

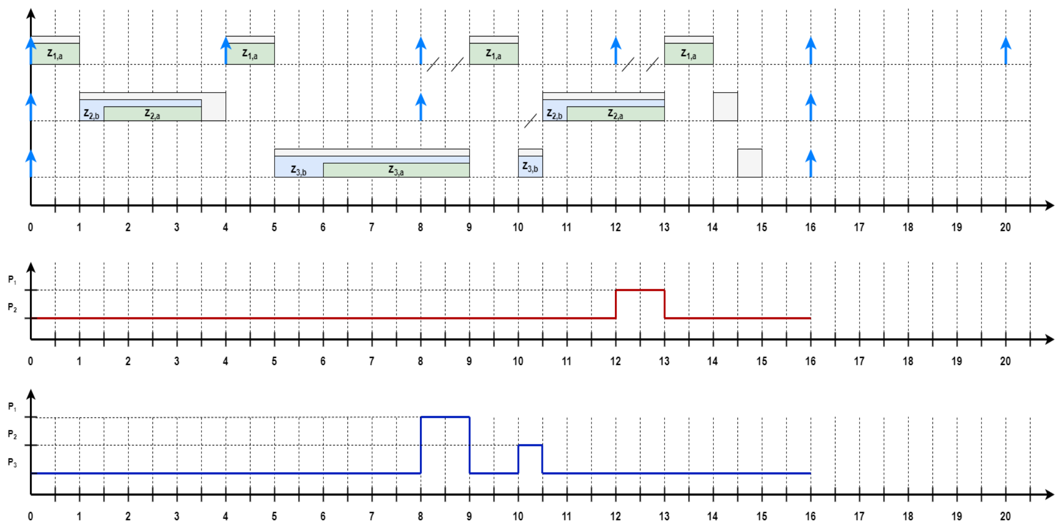
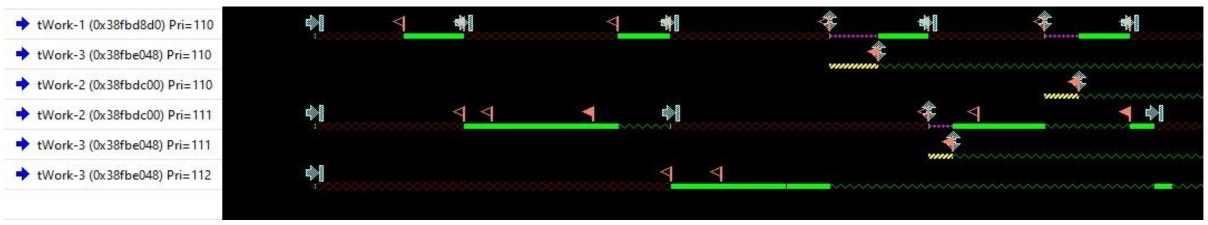


Figure 3.1: Gantt chart of schedule.



tWork-1 (0x38fbd8d0) Pri=110
tWork-3 (0x38fbe048) Pri=110
tWork-2 (0x38fbdc00) Pri=110
tWork-2 (0x38fbdc00) Pri=111
tWork-3 (0x38fbe048) Pri=111
tWork-3 (0x38fbe048) Pri=112

Figure 3.2: Gantt chart from VxWorks.

The Hyper-period of task set :

$$H = l.c.m = \{T_1, T_2, T_3\} = \{4, 8, 16\} = 16$$

Being a simply periodic task set, all tasks are in harmonic relation and synchronized, so if our task set is feasible in the Hyper-period then it can be feasible everywhere. Observing the Gantt chart and the scheduling obtained from VxWorkS in figure 3.2 we can see as first thing that the two Gantt charts are not exactly identical; this is because the task set is run on a virtual machine which emulates VxWorks, so since windows runs the virtual machine as a process we can have non-determinism problems. In any case it is important to notice that there are no missed deadlines and therefore we can conclude that the task set is feasible with the PIP protocol. Being the feasibility analysis only a sufficient and not necessary condition for verifying the feasibility of a task set allow us to confirm that also if the previous tests failed the task set can be feasible. In conclusion, since the tasks are synchronuous we are in the worst possible case for feasibility. This means that if we schedule the same task set by shifting the phases between the three tasks we can say that the task set can be still feasibile.

Chapter 4

Modified/added Code

To build our application, we have started from the files provided in the Laboratory Section RT Lab 6: Resource access protocols (part 2). These files are *metascheduler.c*, *metascheduler.h*, *simulation.c*, *simulation.h*, *resources.c*, *resources.h*, *applications.c*, *applications.h*, *dummyTask.c*, *dummyTask.h*. To correctly build your application the following pieces of code have been modified/added:

- application.c file
- application.h file
- simulation.c file

Code 4.1: Changes made to the code of the application.c file.

```c
/* Application Assignment*/
void ApplicationAssignment(int task) {
  /* one unit product OPMSEC coincides with almost 20ms of
    computation*/
  int unit=40;
  switch (task) {
  case 0: /* task 1: [[Ra:1]] */
      EntrySection(0,0);
      get_busy(unit*OPSMSEC);
      ExitSection(0,0);
    break;
  case 1: /* task 2: [[Rb:0.5 [Ra:2] :0.5] */
      EntrySection(1,1);
      get_busy(0.5*unit*OPSMSEC);
      EntrySection(0,1);
       get_busy(2*unit*OPSMSEC);
      ExitSection(0,1);
      ExitSection(1,1);
    get_busy(0.5*unit*OPSMSEC);
    break;
  case 2: /* task 3: [[Rb:1 [Ra:3]:0.5]:0.5] */
```

```
    EntrySection(1,2);
      get_busy(unit*OPSMSEC);
      EntrySection(0,2);
        get_busy(3*unit*OPSMSEC);
      ExitSection(0,2);
      get_busy(0.5*unit*OPSMSEC);
    ExitSection(1,2);
    get_busy(0.5*unit*OPSMSEC);
    break;
  }
}
```

Code 4.2: Changes made to the code of the application.h file.

```
/* --> To select our application among those available <-- */
#define APPLICATION_ASSIGNMENT

/* .... */

#ifdef APPLICATION_ASSIGNMENT
  #define N_TASKS 3
  #define TASK_PERIODS {4,8,16}
  #define TASK_PHASES {0,0,0}
  #define N_SEMAPHORES 2
  #define CEILINGS { TOP_PRIORITY, TOP_PRIORITY+1 }
  #define APPLICATION ApplicationAssignment
  #define CLEAN_UP_STRING "Simulation ended. Response times:
    %d, %d, %d. Overruns: %d, %d, %d.\n"
  #define CLEAN_UP_PARAMS (_Vx_usr_arg_t) ResponseTime[0], (
    _Vx_usr_arg_t) ResponseTime[1], (_Vx_usr_arg_t)
    ResponseTime[2], (_Vx_usr_arg_t) Overruns[0],  (
    _Vx_usr_arg_t) Overruns[1], (_Vx_usr_arg_t) Overruns[2]
#endif
```

Code 4.3: Changes made to the code of the simulation.c file.

```
void StartSimulation(int sec) {
  static int WorkerCounter = 1;
  int semaphore, SemOptions=SEM_Q_PRIORITY;
  int task;
  char task_name[15][N_TASKS];

  /* initialize simulation parameters */
  for(task=0;task<N_TASKS;task++) {
    NotFinished[task]=FALSE;
    Overruns[task]=0;
    Instance[task]=0;
    ResponseTime[task]=0;
    JobReleaseCounter[task]=1+(Phased?Phase[task]:0);
  }

  /* initialize semaphores */

  if(ResourceAccessProtocol==PIP)
```

```
    SemOptions|=SEM_INVERSION_SAFE;

  for(semaphore=0;semaphore<N_SEMAPHORES;semaphore++)
    Semaphore[semaphore]=semMCreate(SemOptions);
  for(task=0;task<N_TASKS;task++) {
    ResourceCount[task]=0;
    CurrentPriority[0][task]=TOP_PRIORITY+task;
  }
```