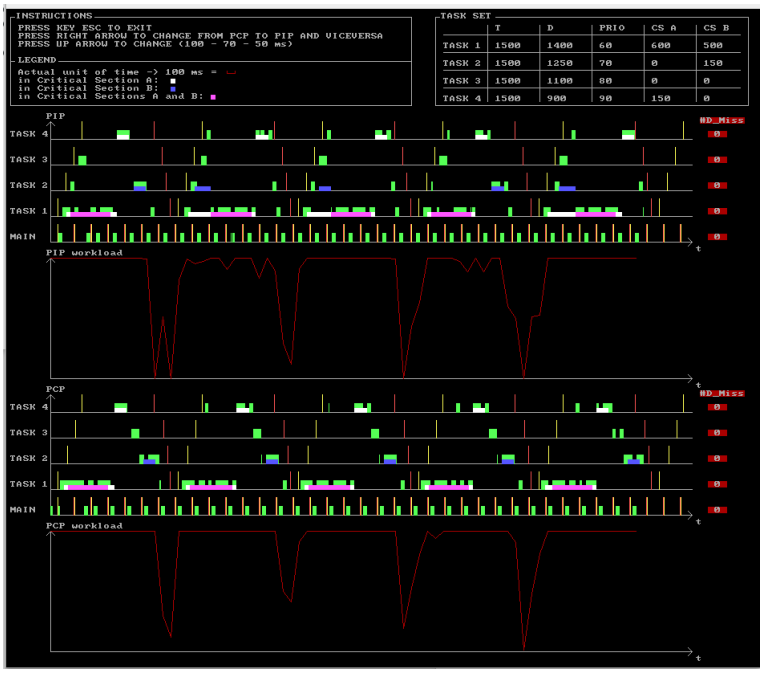
Embedded Computing Systems

Schedule



Veronica Rispo (464056)
veronica_rispo@hotmail.it

February 21, 2017

Contents

1 Introduction

In this report will be presented the most important aspect of the design and the develop of real-time software for a task set in which priority inversion occurs.

1.1 Project Assignment

Visualize the execution of a task set (including the main) with an adjustable time scale. Visualize activation times, deadlines, critical sections with different colors. Build a task set in which priority inversion occurs. Then run the set using PIP and PCP to show that priority inversion disappears. Also visualize the instantaneous workload as function of time.

2 Project Overview

The program shows the behavior of a task set which manages the shared resource access in two possible ways:

- PIP mode (**P**riority **I**nheritance **P**rotocol) : avoids unbounded priority inversion by modifying the priority of those tasks that cause blocking.
- PCP mode (**P**riority **C**eiling **P**rotocol) : the basic idea of this method is to extend PIP with a rule for granting a lock request on a free semaphore. To avoid multiple blocking, this rule does not allow a task to enter into a critical section where there are locked semaphores that could block it.

Every task in the set is customized passing to the program a text file, in which the user can specify the period, deadline, priority, total computational time and the computational time for the critical sections that use the first and the second resource and if the two critical sections are nested.

3 Design

3.1 Task

The execution of the program is composed by many periodic tasks. There are four different types of periodic tasks.

3.1.1 Graphic Task

There is only one graphic task. Its period is equal to the time scale. This task draws in the unit of time which of the tasks (main and generic tasks) has gone running, which of the tasks owns the lock of every critical resource and the workload of the task set. The priority of this task is maximum and uses a different CPU compared to the task set (main and generic tasks).

3.1.2 Generic Tasks

There are four generic tasks. The parameter for these four generic tasks are set by the user.

The parameter that the user must choose and set are:

- Period
- Deadline
- Priority
- Total Computational time

- Computational time in the critical section a
- Computational time in the critical section b
- Nested : if the two critical section are nested

Every tasks of this type work on the same CPU (that is equal to the one used by the main).

3.1.3 Main Task

There is only one main task. Its deadline is slightly lower than its period. This task periodically controls if any key has been pressed and if this key corresponds to a user command and finally if is valid it will do what has been requested by the command itself. This task works on the same CPU of the generic tasks.

3.1.4 Workload Task

There is only one workload task. Its period is equal to its deadline and it is 1ms because this task has to monitor the instantaneous workload in function of time. This task also maintains trace of the task using the CPU and of the task which owns critical resources every ms.

3.2 User Interface

Figure 1: User Interface

At the top of the user interface, there are two elements: on the left the instructions and general information about the program and on the right a table with the information about the four generic tasks set by the user in the text file passed by command line when the program is called. The instructions show to the user which keys have to be used to change the mode (PIP or PCP), to change the time scale, to stop the execution of a set or quit the program.

The bottom part is divided into two horizontal part . The two parts are equal, containing two graphics: one for the use of the CPU and the critical section and the other one for the workload. The upper part is used to show the behavior of the task set when it uses the PIP mode while the lowest one shows the behavior of the task set when it uses the PCP mode.

On the right side of the graphics showing the use of the CPU and the critical section, for each generic task there is a counter which highlights every missed deadline.

4 Implementation

In this section is generically explained the implementation of the principal parts of the program.

4.1 Tasks

The tasks are all real time tasks scheduled according to the Round Robin policy and are all created detached.

4.1.1 Graphic Task

The graphic task has a period equal to the actual time scale value and a deadline smaller (at least half) than the period itself.

The Graphic Task behaves as follows:

- Computes the workload and draw the value in the right graphic (PIP or PCP workload graphic)
- Calls the function that calculates and draws the use of the CPU in the last period in the right graphic
- Calls the function that draws the owner of the lock in the last period in the right graphic
- Finally if we are at the end of the x-axes it cancels the graphic and draws an empty graphic again

4.1.2 Generic Tasks

There are four generic tasks. The parameters of this type of tasks is set by the user in a text file. The parameters are stored in some vectors and initialized by a function called by the main which also controls if the values in the file are correct.

The generic tasks body can be represented in general as the following figure shows:

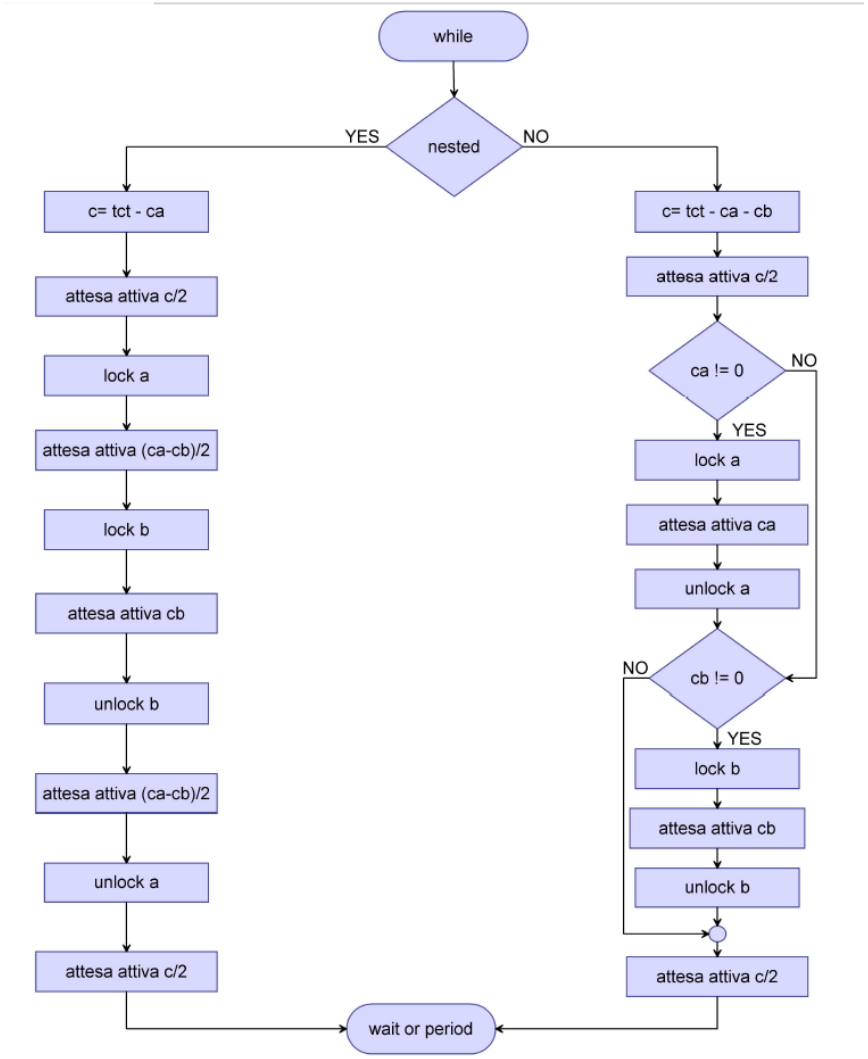


Figure 2: Flowchart

- tct: stands for total computational time
- ca: stands for computational time in the critical section a
- cb: stands for computational time in the critical section b

4.1.3 Main Task

Initially it creates the generic tasks and periodically it controls if any key has been pressed by the analysis_key() function. The function itself manages the user command through a switch.

4.1.4 Workload Task

The Workload Task period and the deadline are equal for this task and correspond to 1 ms.

Each millisecond behaves as follows:

- If no task (generic and/or main) uses the CPU it increments a variable representing that the CPU has not been used in that ms. Later this variable is used in the graphic task to compute the actual workload.
- In a vector the Workload Task stores the value of the task is using the CPU or 7 if no task is using the CPU. In another vector the Workload Task stores the value of the task that owns the lock in that 1ms. These two vectors are used finally in the graphic task in order to draw the two graphics.

4.2 User Interface

The first time the setup of the user interface is done by some function called by the main.

With the function write_instruction() the upper left box is drawn; this box is updated any time the time scale is changed because it contains the information about the actual scale.

The function draw_taskset_box() it is feasible to draw the upper right box containing the information about the parameter set for the tasks.

The lowest part is drawn by the function setup_graphic() one graphic a time. This part is updated whenever the space on the axes is over or if the task set is restarted.

4.2.1 User Commands

With the function analysis_key() called periodically by the main we can analyze the command insert by the user, in particular:

- Key ESC: close the entire program
- Key SPACE: If the task set is running, it stops its execution; otherwise it restarts the task set execution
- Key RIGHT: It stops the task set if it is running and after it changes the mode (if so it was PIP it would become a PCP or vice versa) and it starts the task set using the new mode again.
- Key UP: It changes the actual time scale. There are three possible time scales, that are saved in a vector: 100, 70 or 50 ms. This vector is considered like a circular vector using a variable to keep a track of the position.

4.3 Shared Resource

At each critical resource presented in the program a semaphore of mutual exclusion has been associated to avoid the writing or reading of inconsistent data.

The critical resources protected by the semaphore are:

- **free_ms**, **wl**, **pwl**: these variables are used to compute the instantaneous workload. The workload task, the graphic task and by the analysis_key() function called by the main task could access to these resources simultaneously. The critical sections that works on these variables are protected by the **wlvariablesem** semaphore for mutual exclusion.

- **task[ ]**, **r_task[ ]**, **a_occupation[ ]**, **b_occupation[ ]**: these vectors are used to trace the use of CPU and the owner of the critical section a and b in the period of time (equal to the actual time scale). The graphic task and the workload task could access to these resources simultaneously. The critical sections that works on these vectors are protected by the **mandrawsem** semaphore for mutual exclusion.

- **a**, **b**: these two variables represent the two critical sections that generic task compete for. Those critical sections are protected by two mutex semaphores that use the protocol PIP or PCP according to the current active mode. The mutexes are: **mux_a_pip**, **mux_b_pip**, **mux_a_pcp** and **mux_b_pcp**.

4.4 File Content

- taskRT.h and taskRT.c : They contain the declaration and the definition of the structures and the functions dealing with the handle and the creation of threads using the pthread library.

- timeplus.h and timeplus.c : They contain the declaration and the definition of the structures and the functions dealing with the handle of time using the time library.

- schedule.c : It contains the main function and the principal function of the program.

4.4.1 Task parameter file

The text file must be passed by the command line when the program is called.

The text file for setting the task parameter must be formatted in this way:

- Every row represents a task
- There must be four rows
- In each row,there must be seven elements
- In order:
  - Period
  - Deadline
  - Priority
  - Total Computational time
  - Computational time in the critical section a
  - Computational time in the critical section b
  - Nested : if the two critical section are nested

The values in each row must respect this rule:

- If nested is set to 1, it means that it is true:
  - The two computational time for the two critical section must be different from zero
  - The computational times for the critical section **b** must be less than the one for the critical section **a** because it is nested in the second one
  - The total computational time must be greater than the computational time for the critical section **a** (that includes the one for the critical section **b**)
- If nested is set to 0, it means that it is false and the total computational time must be less than the sum of the computational time of the two critical sections

For example, the file could be:

```
1200 1000 60 800 600 400 1
1200 1000 70 500 0 300 0
1200 1000 80 300 0 0 0
1200 1000 90 500 300 0 0
```

5 Test

Changing the text file passed by the command line when the program is called the user can do the different testing configurations of the task set and see their behavior.

In every test case the Task #1 is created immediately, then 50ms after the Task #2 is created, after 50ms the Task #3 is created and at the end after 50ms the Task #4 is created.

5.1 Test #1

For the first test the file **sharedfeasible.txt** has been set as the figure below shows:

	Period	Deadline	Priority	Comp_Time	Sect_a_time	Sect_b_time	Nested
Task #1	1500	1400	60	700	600	500	1
Task #2	1500	1250	70	250	0	150	0
Task #3	1500	1100	80	100	0	0	0
Task #4	1500	900	90	250	150	0	0

Figure 3: Test #1 task set

With this configuration of the task parameter we have the result showed in the following figure.

Figure 4: Test #1 Graphics

As we can see from the figure the deadlines are respected with this configuration of the task parameter.

5.2 Test #2

For the second test the file **sharednotfeasible.txt** has been set as the figure below shows:

	Period	Deadline	Priority	Comp_Time	Sect_a_time	Sect_b_time	Nested
Task #1	1500	1400	60	700	600	500	1
Task #2	1500	1250	70	250	0	150	0
Task #3	1500	1100	80	100	0	0	0
Task #4	1500	900	90	450	200	100	0

Figure 5: Test #2 task set

With this configuration of the task parameter we have the result showed in the following figure.

Figure 6: Test #2 Graphics

As we can see from the figure the deadlines are not respected with this configuration of the task parameter.