RTVideo : Project Report

Year 2015/2016

Matteo Rotundo

Contents

1 Introduction

1.1 Assignment Details

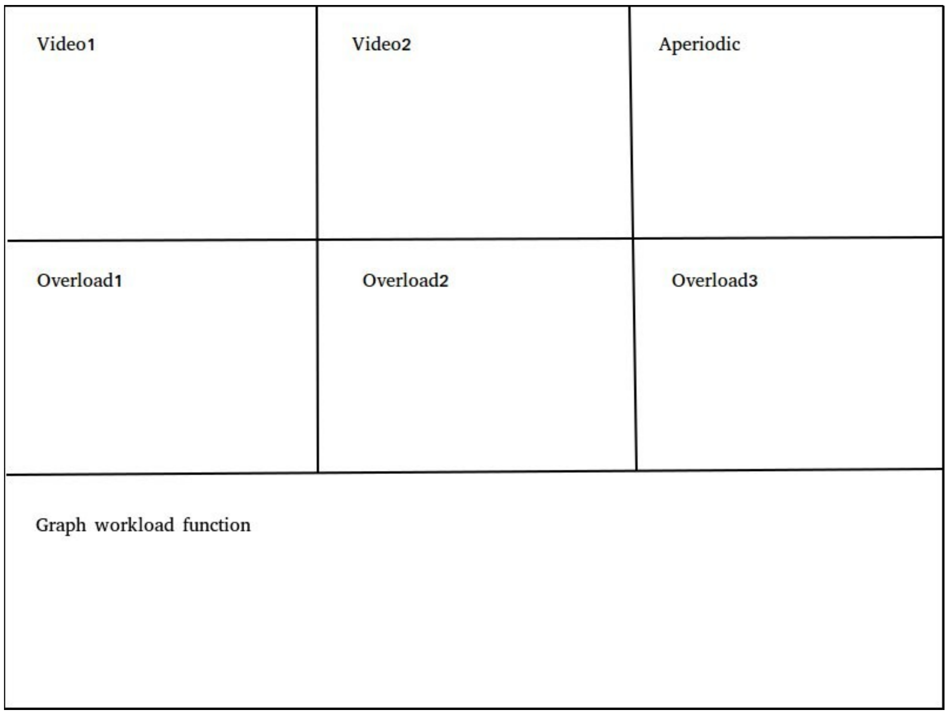
The project has to visualize the execution of videos for monitoring the difference of performance with different schedulers. It must be implemented a way to create an overload condition for the tasks that play videos for analyzing their response to critical situations. Besides, it must be visualized on screen an aperiodic job that handles mouse events and the graph of the actual workload. It must be possible to select a different policy of scheduling for the tasks. In particular, it should be used the following Linux schedulers:

- SCHED_FIFO;
- SCHED_DEADLINE;

The project should be implemented in a Linux environment, using Allegro and Pthread libraries.

1.2 Basic functionality idea

The basic graphics idea is the creation of an interface following the scheme shown in figure.



The project can be executed from the user with the two different schedulers by passing to the application the value of the policy in Linux.

SCHEDULER	VALUE_OPT
SCHED_FIFO	1
SCHED_DEADLINE	6

An example of a command to execute the application with SCHED_DEADLINE:

```
./RTVideo -s 6
```

2 Design Project

The basic idea is to arrange the task-set with :

- Two tasks that have to play a video each one (Player tasks);
- Two tasks with one that checks mouse events and eventually awakes the other task (Aperiodic task) ;
- A task that reads from keyboard and activates eventually an overload task;
- Three tasks that must draw an animation on the screen and create an overload condition for the player tasks (Overload tasks);
- A task that must plot online a workload function on screen;

CPU Affinity

In order to guarantee that overload tasks affect player tasks, it is needed an affinity to a specific core by all the tasks. Otherwise, it is possible that a task could be scheduled on a different unit of cpu preventing any attempt of overload.

Priority

The assignment of priority is equal for all the tasks (HIGH PRIORITY), except for the plot task that has low priority because it must execute only if no other task is in execution. (it counts the idle time).

2.1 Environment choices

Linux sets by default 95% of bandwidth to real-time task and the rest to other tasks scheduled by different class of scheduling. The task-set must be isolated from the system in all the cases. In particular, the plot task must not be preempted from other tasks in the system, so it is needed to set all the bandwidth to the class real-time with this command:

```
>> sudo echo -1 > /proc/sys/kernel/sched_rt_runtime_us
```

Another problem that could affect the correct behavior of the application, is the frequency of CPU. In fact if it is not stable, some jobs could be executed quickly than other giving the wrong conclusion that the workload changes due to tasks itself.

In order to avoid this situation, it's useful a tool like indicator-cpufreq. The command for installing is:

```
>> sudo apt-get install indicator-cpufreq
```

With latest versions of Ubuntu (after 14.10), is not possible to set a specific frequency. For this purpose, after installing the tool must be changes some things in the grub. Here the following instructions for this:

```
>> cd /etc/default
>> sudo cp grub grub.backup
>> sudo nano grub
```

Then at the line beginning with GRUB_CMDLINE_LINUX_DEFAULT it has to be appended the words intel_pstate=disable before the final quote. In the end:

```
>> sudo update-grub
>> sudo reboot
```

3 Implementation

The whole source code has been divided in smaller modules with each one its functionality:

- RTVideo.c contains the main e all the initialization before start the application;
- In Draw.h and Draw.c there are all of constants and functions useful for drawing on the screen;
- In Task.h and Task.c there are the structures and the body of all the tasks;
- In Periodicity.h and Periodicity.c there are the function for handling periodic tasks and time;
- In Sched_new.h and Sched_new.c are implemented the functions for scheduling and for setting the affinity;

3.1 Modules

In this chapter it will be explained every modules of the whole project and how they are implemented.

3.1.1 RTVideo

This module is the main of the program. First the main gets and checks the policy that must be set, then it starts a calibration. This consists in executing a task like the plot task to count in order to get a value of counting of the machine where the application is executed. It's needed in order to guarantee the portability of the application.

It initializes all the stuff for Allegro and draws the interface on the screen. After that, it initializes all the structures setting all the parameters useful for the tasks like priority, period and deadline. Finally it creates the tasks, except overload tasks, and joins on them.

3.1.2 Periodicity

This module provides the basic data structure and functions needed by periodic tasks. It provides time manipulations, deadline miss checking, next activation time computation and a function which puts the thread to sleep until the next activation.

With SCHED_DEADLINE is different handling periodic task, because if we use clock_nanosleep there's a phase displacement between the server and the task scheduled. In order to avoid this, the function that implements the wait until the next period checks if the policy is SCHED_DEADLINE and then calls the sched_yield instead of the clock_nanosleep. The function sched_yield resets the actual budget, so the tasks can synchronize with the server.

Due to this, the value of deadline miss that it's checked in the application with SCHED_DEADLINE is meaningless.

3.1.3 Task

In this module there are all the body of tasks. All the threads set the scheduling policy and the affinity before starting their jobs. In order to avoid starvation, it is needed a barrier after these initial operations. In fact, the plot task must be scheduled with SCHED_FIFO and because sched_rt has set bandwidth 100% it is possible that a task temporarily scheduled with SCHED_OTHER can't go in execution.

Play Tasks

The two player task must be implemented like periodic tasks with a period given by the inverse of the frame-rate of the each video. So they play videos loading a frame with a blit every period.

Activation task

This task reads from keyboard and activates eventually, if the key pressed is ENTER, an overload task. If the key pressed is EXIT it sets a variable to notify to all the tasks that they must terminate. This operation shuts down the application because the main is joining on the tasks.

Mouse Task

The management of the mouse is not blocking, so for creating an aperiodic task it is needed another task. This checks periodically mouse events with a period given by the inverse of the default polling rate of the mouse. It awakes the aperiodic task that draws an ellipse animation with random color on screen. Drawing an animation is not a computation that can create overload, thus the aperiodic task has to perform high computational activities in addiction to its main work.

Overload tasks

With the same basic idea just explained the overload tasks do a busy wait in addiction the selected animation, but in order to create an high load they perform their action periodically. The animations of these tasks are:

- A ball that bounces from a point of a screen to another;
- A blinking circle;
- An animation pseudo-clock;

Plot Task

The plot task must draw the workload function of the task-set. For this purpose, when it is in execution it counts for a fake period in order to compute the idle time (we call this value n). So the workload is computed by the following formula:

$$Workload = 1 - n / N$$

which N is the value of the counting variable in a situation where the plot task is alone. This value is computed in the phase of calibration mentioned earlier.

3.1.4 Draw

This modules is a library for drawing the interface. It provides useful constants like where are located the sub-screens of the interface or the scales used for the cardinal graph. Besides constants, it provides all the function for handling the graph. In particular:

```
void draw_axisX(int, int);  // This function draws the axis X of our graph
void draw_axisY(int, int);  // This function draws the axis Y of our graph
void clean_graph();     // This function cleans the graph for restarting to draw
void draw_point(int x, int y); // This function draws a point in the graphics
```

The scale used for axis x is that every 4 pixels there are 150ms (namely the fake period of the plot task). For the axis y instead the scale is that every 2 pixel there's 1% of workload.

3.1.5 Sched_new

This is is the module where it's implemented the function for setting the scheduler. It's needed because the pthread library does not support SCHED_DEADLINE.

In addiction, it's implemented the function for setting the affinity with the CPU. It is not possibile to use other functions given by standard libraries because they are not compatible with SCHED_DEADLINE.

Before the affinity it is needed a setup that it is encapsulated in the function *setup_affinity_folder()* that consists in :

- Creating a folder in “/sys/fs/cgroup/cpuset” for the task-set;
- Updating the memory nodes;
- Sets in “/sys/fs/cgroup/cpuset/coyset.cpus” which CPU will be used by the task;

The function *set_affinity()* writes in “/sys/fs/cgroup/cpuset/taskset” the tid of the calling thread in order to set the affinity with the selected CPU.