Assignment 3. Real-time Tasks Models in Linux (100 points)

Assignment Objectives:

1. To program real-time tasks on Linux environment, including periodic and aporadic tasks, event handling, priority inheritance, etc.
2. To use Linux trace tools to view and analyze real-time scheduling.

Assignment:

As shown in the following diagram, periodic and aporadic tasks in real-time systems can be simply expressed as endless loops with time- and event-based triggers. In the task body, specific computation should be done and locks must be acquired when entering any critical sections.

```
TASK periodic_task()
{
        < local variables >
        initialization() and wait_for_activation();
        while (condition) {
                <task body>
                wait_for_period();
        }
}
```

```
TASK aperiodic_task()
{
        < local variables >
        Initialization() and wait_for_activation();
        wait_for_event();
        while (condition) {
                <task body>
                wait_for_event();
        }
}
```

(from "*Ptask: an Educational C Library for Programming Real-Time Systems on Linux*"
By Giorgio Buttazzo and Giuseppe Lipari, *EFTA 2013*)

In this assignment, you are asked to develop a program that uses POSIX threads to implement these task models on Linux environment. The task body is defined in the following BNF:

<task_body> ::= <compute> { < CS> <compute> }

<CS> ::= <lock_*m*> <compute> {<CS> <compute>} <unlock_*m* >

where "lock_*m*" and "unlock_*m*" are locking and unlocking operations on mutex *m*, and "compute" indicates a local computation. To simulate a local computation, we will use a busy loop of *x* iterations in the assignment, i.e.

```
int i, j=0;
for (i = 0; i < x; i++)
{
        j = j + i;
}
```

The input to your program is a specification of a task set which is shown in the following example:

```
line 1:  5 3000                          // there are 3 tasks and the execution terminates after 3000ms
line 2:  P 20 500 200 L3 300 U3 400      // task 1 is a periodic task of priority 20 with period 500ms.
                                         // In its task body, it runs the busy loop for 200 iterations, locks
                                         // mutex 3, runs the busy loop for 300 iterations, unlocks
                                         // mutex 3, and finally runs 400 iterations of the busy loop.
line 3: P 10 200 200 L2 100 L3 500 U3 400 U2 200
                                         // task 3 invokes nested locking
line 4: P 30 750 100 L3 600 U3 100       //
```

```
line 5: A 18 1 500          // task 4 is an aperiodic task of priority 10. It is triggered by
                            // event 1 and then runs 500 iterations of the busy loop.
line 6: A 60 0 500          // task 5
```

To avoid the job of parsing input specifications, you may consider a fixed specification of the 5 task set listed in the example for the assignment. Your program should create a thread for each task to perform task operations given in the specification. Additional requirements of the program are:

1. At most 10 mutex locks are needed for shared resources and two external events are considered. Events 0 to 1 arrive when we release left or right mouse buttons, respectively. Also, an event may trigger the execution of multiple aperiodic tasks.
2. When the current iteration of a periodic task is not done before the end of the period, its next iteration should be started immediately as soon as the task finishes its current task body.
3. The priority numbers given in input files are real-time priority levels and tasks are scheduled under the real-time policy SCHED_FIFO.
4. All tasks should be activated at the same time. When the execution terminates, any waiting tasks (wait_for_period or wait_for_event) should exit immediately. Any running or ready task should exit after completing the current iteration of its task body.
5. Using conditional directive, your program can invoke normal and PI-enabled pthread mutexes for locking resources.
6. If your host is with multiple cores, you should use CPU affinity to ensure all threads are bound to one core.

To verify the scheduling events of the tasks in the host system, you can use “trace_cmd” to collect schedule events, such as context switch and wakeup, (or any other useful events) from the Linux internal tracer ftrace. The traced records can then be viewed via a GUI front end kernelshark.

Along with the program, you are required to prepare a report. In the report, you need to verify the scheduling of the tasks a specific task set by showing that

1. Tasks are scheduled under the SCHED_FIFO with the assigned priorities, and
2. Priority inversion under normal and PI-enabled mutexes.

Here are some suggestions you can consider:

- You may want to develop two generic task functions (for periodic and aperiodic respectively) which take the specification of task body, and period or event, to perform task execution. In the main program, threads can be created with proper priorities and then call the task function with the corresponding parameters. Note that these two generic task functions must be reentrant.
- You may need a separate thread to read in mouse events from the device “evdev” and dispatch the events to the destination aperiodic tasks.

Due Date

The due date is set to 11:59pm, July 12, tentatively.

What to Turn in for Grading

- Create a working directory to include your source files (.c and .h), makefile(s), readme file and your report (in pdf format). Compress the directory into a zip archive file named RTES-teamX-assgn03.zip. Note that any object code or temporary build files should not be included in the submission. Email the zip archive to the instructor by the due date and time.
- Please make sure that you comment the source files properly and the readme file includes a description about how to make and use your software. Don’t forget to add each team member’s name and student id in the readme file.

- There will be 20 points penalty per day if the submission is late. If you have multiple submissions, only the newest one will be accepted. If needed, you can send an email to the instructor and TA to drop a submission.
- Your team must work on the assignment without any help from other teams and is responsible to the submission. No collaboration between teams is allowed, except the open discussion in the class.
- Here are few general rule for deductions:
    - No make file or compilation error -- 0 point for the part of the assignment.
    - Must have "–Wall" flag for compilation -- 5-point deduction for each warning.
    - 10-point deduction if no compilation or execution instruction in README file.
    - Source programs are not commented properly -- 10-20-point deduction.