Concepts and Software Design for CPS

Lab 5: POSIX Timers, Signals and Real-time Scheduling

Raphael Trumpp Binqi Sun

Chair of Cyber-Physical Systems in Production Engineering
Technical University of Munich
Munich, Germany

Additional Material

You can find related material at the following sources:

- Lecture 7
- Lecture 8
- Lecture 9
- O'Reilly, Posix 4, Programming for The Real World
- (German) Openbook Linux-UNIX-Programmierung

Assignment 5 (due: 13.01.2022)

Server

Not all operating systems are POSIX compliant. For this assignment we provide you with a virtual server which you can use when connected via eduroam or VPN.
To access it, use an SSH terminal of your choice, e.g.,

- on Linux: openssh client
  `(Ubuntu) sudo apt install openssh-client`
- on OS X: openssh client
  `https://osxdaily.com/2017/04/28/howto-ssh-client-mac/`
- on Windows: putty
  `https://www.putty.org`
  To paste text: Shift+Ins (de: Shift+Einfg)

Usage: `ssh -p 22xy root@10.162.12.150`
Port: 22xy (where x is your session number and y is your group number)
Username: root
IP address/Host: 10.162.12.150
Password: (you can find it in moodle)

Server: Commands

The command line program is called Bash (in case you want to search for some more commands).

Here are some commands you might need:

pwd	print working directory (where you currently are)
ls	list files and directories
cd	change to another directory (e.g., `cd /root/`)
mkdir	make/create directory
exit	close connection to server
ctrl-c	(key press) stops the running command (sends SIGINT)
cat	dump file content to the terminal (e.g., `cat hello_world.c`)
man	show manual page (e.g., `man timer_create`)

Some more are: `mv` = move, `rm` = remove, `rmdir`, `clear`, ...
You can work in the folder `/root` or create a subfolder for this assigment.

Server: File Transfer

You don't have to edit source code on the server. Instead you can work on your PC and transfer files to the server via SFTP.

For all operating systems you can use FileZilla:
`https://filezilla-project.org/`

Use: File > Site Manager > New Site
Then use the parameters from above (additionally: Protocol = SFTP, Logon Type = Ask for password).

Once connected you will see your computer's files on the left and the server's files on the right.

The servers are a temporary environment, don't store any important information on them. Also, don't use them for other purposes than the lab work.

Working on the Server

For this assignment we will use the GNU C Compiler (short: `gcc`). When invoked with a C file, e.g.,
`gcc hello_world.c`
it will generate an executable binary called `a.out`.

You can execute this binary by prepending it with the relative path:
`./a.out`

If you want to execute the binary on a particular core use *taskset*:
`taskset 0x01 ./a.out`
Here a.out will be executed on core 0 (as only bit 0 is set). For the assignment, please use your group's assigned core (see moodle).

For this assignment we will use the compiler with these options:
`gcc -Wall -o myprogram hello_world.c -lrt`

They are: `-Wall` = show all warnings, `-o` = choose a binary name other than a.out, `-lrt` = link with posix real time library.

Task 1: Periodic Task

Write a periodic real-time task that uses a POSIX timer and the signal it sends to implement a periodic task.

Use at least the following POSIX functions:

- `timer_create` (clockid = `CLOCK_REALTIME`)
- `timer_set_time`
- `sigaction` (signum = `SIGALRM`)
- `sigwait` (signum = `SIGALRM`)

Your task should output a warning if the deadline (equal to period) is missed. Hint: If a signal is received and the process is not blocking on sigwait, the registered signal handler is used instead.

Notes: To provoke a deadline miss you can use the function `sleep`. You can look up POSIX function references using `man` (e.g., `man sigwait`) on the server.

Task 2: Task Scheduling

Extend the periodic task from the previous assignment by adding the following command line arguments:

- period (in milliseconds, from 1 ms to 999 ms)
- priority
- CPU load factor
- scheduling policy (*SCHED_FIFO* or *SCHED_RR*)

Please check the Posix example from the Lecture 9

Task 2: Task Scheduling

Every period, the task, after receiving the timer signal, should:

- execute a generic load:

```
for(int i=0; i<load*1000; i++){
    /* do nothing, keep counting */
}
```

  where load is the command line parameter,

- print the response time (i.e., the time period between receiving the timer signal and computation end)[1].

[1] E.g., *timer_gettime* gives you a duration relative to the timer expiration

Task 2: Task Scheduling

Run two instances of your program on the same processor:

- open two SSH connections to the server
- use *taskset* command to assign the same processor

Try the following parameters:

	period	*load*
task 1	500 ms	100000
task 2	550 ms	100000

Task 2: Task Scheduling

First, observe the response time of each task executed in isolation (without co-runner). Then, execute both tasks together in the following configurations:

- both tasks use *SCHED_RR* and have the same priority
- both tasks use *SCHED_RR* and task 1 has higher priority

Compare the observed response times. Explain the differences. Is the response time of the lower priority task always the same?

Next Lab: Lab 6 on 13.01.2022

Next week: Q&A Meetings!

Next Lab: Lab 6 on 13.01.2022

- Topic: POSIX - Message Queues & Synchronization
- **Feedback meeting: Assignment 5**
- Hand out Assignment 6