EURECOM
Sophia Antipolis

Exam
Operating Systems - OS - Fall2018

Ludovic Apvrille
ludovic.apvrille@telecom-paristech.fr

February, 2019

Authorized documents: Nothing! The grading takes into account the fact that you don't have any documents with you.
A grade is provided for each question (beware: be sure to organize your time with regards to the grading policy). 1 additional point is given for general appreciation, including writing skills and readability.

1 Course knowledge (6 points, ∼40 minutes)

a. Give the example of a blocking system call, and define what "blocking" means. Would it be possible to implement a non blocking version of this syscall? Quickly sketch how you would implement this syscall or why this would not be possible. [3 points]

b. Could you explain when "condition variables" are necessary? Do use an example to compare two different programs which have a similar behaviour, but where only one of them is able to use a condition variable. Said differently, explain how you can replace condition variables with a similar code. [3 points]

2 Real-Time Linux (13 points, ∼80 minutes)

a. Cite three differences — other than scheduling — between Operating Systems and Real-Time Operating Systems. [1.5 points]

b. The documentation of Xenomai[1] states:

[1]Memo: Xenomai is a real time version of Linux

To get your application threads to be really considered as real-time threads by Xenomai scheduler, you will have to get them to use the real-time scheduling policy (called SCHED_FIFO). Note, however, what the SCHED_FIFO scheduling policy means: it means that the scheduler will run a thread with this scheduling policy as long as it is runnable, and no other thread of higher priority is runnable.

Describe in your own words - e.g. using a figure - how this scheduling policy works. Is it able to handle all (urgent) timing constraints such as close deadlines? If not, how could it be improved? [2 points]

c. In Xenomai, there is a timer function *xntimer_get_overruns()*. Its documentation is the following one:

```
xntimer_get_overruns()

This service returns the count of pending overruns for the last tick
of a given timer, as measured by the difference between the expected
expiry date of the timer and the date now passed as argument.

Parameters
    timer The address of a valid timer descriptor.
    now current date (as xnclock_read_raw(xntimer_clock(timer)))

Returns
    the number of overruns of timer at date now
```

The documentation of POSIX for timer overruns is as follows:

Under POSIX.1b, timer expiration signals for a specific timer are not queued to the process. If multiple timers are due to expire at the same time, or a periodic timer generates an indeterminate number of signals with each timer request, a number of signals will be sent at essentially the same time. There may be instances where the requesting process can service the signals as fast as they occur, and there may be other situations where there is an overrun of the signals.

The timer_getoverrun function helps track whether or not a signal was delivered to the calling process. DIGITAL UNIX P1003.1b timing functions keep a count of timer expiration signals for each timer created. The timer_getoverrun function returns the counter value for the specified timer ID. If a signal is sent, the overrun count is incremented, even if the signal was not delivered or if it was compressed with another signal. If the signal cannot be delivered to the calling process or if the signal is delayed for some reason, the overrun count contains the number of extra timer expirations that occurred during the delay. A signal may not be delivered

if, for instance, the signal is blocked or the process was not scheduled. Use the timer_getoverrun function to track timer expiration and signal delivery as a means of determining the accuracy or reliability of your application.

If the signal is delivered, the overrun count is set to zero and remains at zero until another overrun occurs.

So, what is meant by "timer overruns"? You may use a figure to explain your points. Also, why is it important when developing a real-time application to take these overruns into account? [3 points]

d. Write a pseudo-C code implementing a strict periodic task using timers, e.g. a task waking up strictly every 5ms. In case you have understood overruns (see previous question), I suggest you enhance your code using *xntimer_get_overruns()* . [4 points + 2.5 points if you take into account overruns]