EURECOM
Sophia Antipolis

Exam
Operating Systems - OS

Ludovic Apvrille
ludovic.apvrille@telecom-paristech.fr

February, 6th, 2015

Authorized documents: Nothing! The grading takes into account the fact that you don't have any document with you.
A grade is provided for every question (beware: do organize your time, e.g., last question is a 4-point question). 1 additional point is given as a general appreciation, including written skills and readability.

1 Course understanding (6 points, ∼30 minutes)

a. Multi-programming enables more than one single process to apparently execute simultaneously. How can an Operating System achieve this on a mono-processor computer? For your answer, clearly state the minimal hardware support that is needed to achieve this. [2 points]

b. Virtual memory is always handled by OS of PCs, but its management is optional in many RTOS. First, recall what is the main purpose of virtual memory. Then, explain why it is not always useful in real-time and embedded systems. [2 points]

c. You have experimented with RTAI that processes/threads/tasks could wake up in advance with regards to their expected wake up time. Explain why this can occur. [2 points]

2 Linux kernel modules (13 points, ∼90 minutes)

If you want to add code to a Linux kernel, a common way to do is to add some source files to the kernel source tree, or to modify existing files, and recompile the kernel, as we have done to add a Linux system call during a lab. But you can also add code to the

Linux kernel while it is running, using a Loadable Kernel Module (LKM). A RTAI task, with which you have played with during the labs, is a LKM.

a. Is it a good idead to use kernel modules to implement the following functions, or is it better to rely on user-level processes: Drivers? Protocol stacks? Scheduler? System calls? ssh server? Closely explain your answer. [1 points]

b. The crash of a kernel module can freeze the whole operating system. Explain why a user-level application should not be able to freeze the entire system, and why a LKM can easily do this. In particular, explain which hardware elements are involved in that difference between user and kernel-level applications, and how they are involved. Last, provide a code that leads the linux kernel to crash in the following *init* function of a module: [3 points]

```
#include <linux/init.h>
#include <linux/module.h>

MODULE_LICENSE("GPL");

static int module_init(void) {
  ...
}
```

c. Kernel modules can also be used to intercept system calls. The basic idea is to replace in the list of pointers to system call functions (this list is an array) the default reference of the system call to intercept with a reference to the function provided by the kernel module. The following code provides a typical way to do so. Explain the purpose of the following code. Also, explain how to execute that code (this is similar to starting a task in RTAI). [3 points]

```
#include <linux/init.h>
#include <linux/module.h>
#include <linux/unistd.h>
#include <asm/arch/unistd.h>

MODULE_LICENSE("GPL");

extern void* sys_call_table[];
int (*original_fork)(struct pt_regs);

static int module_init(void) {
  printk( KERN_ALERT "[edu]   Module successfully loaded\n");
  printk( KERN_ALERT "[edu]   Intercepting fork() syscall... ");

  original_fork = sys_call_table[__NR_fork];
  sys_call_table[__NR_fork] = edu_fork;

  printk( KERN_ALERT "done/n");
  printk( KERN_ALERT "[edu]   Starting Loging system calls\n");
  return 0;
}
```

d. Implement the *edu_fork* system call that performs a *printk* each time the *fork* system call is made in the system. After the *printk*, *edu_fork* should call the default *fork* system call. [3 points]

e. When the module is unloaded, the default *fork* system call should be called instead of *edu_fork*. Provide the implementation of *module_exit()* [1 point]

f. Why isn't it a good idea to allow the interception of system calls with kernel modules? How can this be prevented in the kernel? [2 points]