Course Operating Systems

Final Examination – Fall 2004

February, 2005

Duration: 2h

Ludovic.Apvrille@telecom-paris.fr

Authorized documents are limited to course's slides and to the code you produced during lab sessions. Calculators are authorized but quite useless. Answers should be as concise as possible. Also, you are free to answer either in **English** or in **French,** but please do not mix them!

I. Understanding of the course (4 points)

(a) Closely explain the drawbacks of using the micro-kernel approach when implementing an Operating System?

(b) What is the main interest in using buffers for achieving input/output on devices? Also, what is double buffering? For what kind of devices is it used? Justify your answer.

(c) Describe the fundamentals of process signals under UNIX. What were the issues of applying process signals to threads? How has it been solved?

(d) How are used / implemented interrupt handlers of real-time systems to enhance the efficiency of these systems?

II. Operating systems' reliability (4 points)

The following article has been extracted from LinuxWorld.

> *Linux Kernel 2.6 vs Windows XP: 2.6 Has Far Fewer Bugs*
> *"Well below the industry average for commercial enterprise software," Wired reports*
> *December 15, 2004*

Four years ago at the Stanford University Computer Science Research Center an analysis project began to compare the 5.7 million lines of code in the 2.6 Linux production kernel with the 40 million lines of code in Windows XP.
According to Wired, the resulting study - which appeared yesterday - shows that, while in commercial code the industry standard is that there would be expected to be 114,000 to 171,000 bugs in 5.7 million lines of code, in Linux 2.6 there were just 985 bugs.

"This is a benefit to the Linux development community," Andrew Morton, lead Linux kernel maintainer, told Wired, "and we appreciate Coverity's efforts [Coverity is a software engineering startup that now employs the five researchers who conducted the study] to help us improve the security and stability of Linux."

Of the 985 bugs identified, 627 were in critical parts of the kernel, Wired noted, while "another 569 could cause a system crash, 100 were security holes, and 33 of the bugs could result in less-than-optimal system performance."

(a) Provide a very short summary of this paper (2 lines).

(b) Comment this paper. You should clearly emphasize which issues are raised, how they are solved, and comment whether results are relevant or not.

(c) What would be your approach to you compare the number of bugs in two operating systems?

III. Programming with POSIX (6 points)

The goal of this exercise is for you to propose two different implementations of periodic threads using **POSIX primitives** (and not primitives specific to RT-Linux). More precisely, you should propose a C program running a thread that periodically executes a *foo*() function. Your first implementation should not use timers whereas the second one should make use of them.

(a) Your two implementations face a fundamental issue raised by real-time systems. Explain this issue and explain how you intend to solve it in your implementations.

(b) Provide your first implementation (without timer) and explain it. (For example, you may provide the implementation of a periodic thread calling the function foo() periodically with a period of 1 ms).

(c) Provide your second implementation (based on timers) and explain it.

IV. Scheduling algorithms (6 points)

We consider the following task set composed of 6 tasks P1, P2, P3, P4, P5, P6.
Notations: Ci and Ti designate respectively the capacity and the period of task Pi.

P1: C1 = 10, T1 = 80, the last 3 units of execution time requires exclusive access to a resource (protected by a mutex semaphore S) shared with task P4. P1 is periodic and its relative deadline is equal to 80.

P2: C2 = 20, T2 = 90. P2 is periodic and its relative deadline is equal to 90.

P3: is also periodic but (important !!) is *interrupt-driven*. It means that all processing is done by an interrupt routine and cannot be preempted. C3 = 20, T3 = 150; relative deadline is equal to 150.

P4: C4 = 20, T4 = 200; P4 executes 10 units of time, then requires S for 5 units (in order to access exclusively to the resource shared with P1) then after releasing S, P4 executes another 5 time units. P4 is periodic and its relative deadline is 200.

P5: is a periodic task whose execution is split into 2 parts. The first part is *interrupt-driven (no preemption)* and has a duration of C5a = 5 units of time. The second part is a normal task processing of C5b = 15 units of time. T5 = 250. Relative deadline is 250.

P6: C6 = 30, T6 = 300. P6 is periodic and its relative deadline is equal to 300.

We assume that the priorities are allocated according to the Rate Monotonic method.

Question 1

We consider *only* tasks P1, P2, P4 in the task set (we do not put the other tasks in service).

(a) Determine the worst case scenario of execution for the Task P1 assuming no special algorithm for priority inversion handling.
(b) Determine the worst case scenario of execution for the Task P1 assuming that Priority Inheritance Protocol is used to manage the semaphore S.

For both cases, give the termination time of task P1.

From now, we assume the protocol PIP for managing semaphore S and *all tasks* P1 to P6 are present in the system.

Question 2

For each task P1, P2, P3, P4, P5, P6, give the value of their “blocking factor” B1, B2, B3, B4, B5, B6, if any.

Question 3

Determine if the task set is schedulable under Rate Monotonic. (Use any valid method to justify your answer)

Question 4

Give the worst termination time of task P6.