“Operating Systems” Course

Final Examination – Fall 2006

February, 2007

Duration: 2h

ludovic.apvrille@enst.fr* and *christian.bonnet@eurecom.fr

No document regarding OS or RTOS is allowed. Questions on OS or RTOS do take into account the fact that you don't have any document on that topic. The only documents allowed are the three slide sets and the paper that was given to you during the two following lecture sessions: the 17th of January and the 31st of January. Having other documents (or communicating devices) with you will result in the regular cheating procedure.
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 (5 points)

(a) When a thread or a process waits for the expiration of a timer, it is always awoken later than the expected date. Can you explain what this extra delay is due to? Which techniques are used in real-time operating systems to minimize that delay? Do explain why theses techniques minimize that delay (do not only list them). (3 points)

(b) What is the difference between a system call and a library function? Also, what is the difference between a static library and a dynamic library? (2 points)

II. Real-time and Embedded Applications (4 points)

A real-time and embedded application is usually made of threads implementing various functionalities closely related to the mission of that application, plus one thread- called housekeeping thread- whose purpose is to observe that other threads haven’t crashed (housekeeping).

In this exercise, you are asked to propose a POSIX implementation for the housekeeping thread. I DO NOT ask you to provide a full code, I just ask you to provide a sketch of program, and to comment this program (i.e. explain me how a thread is observed as crashed etc.), and also to answer the following questions:

(a) When is the housekeeping thread started?

(b) What actions can be taken if a thread is identified as crashed? Your answer should take into account whether the crashed thread was storing important data for other threads, or not.

III. Scheduling tasks (4 points)

Consider a set of 3 independent tasks (Task P1, Task P2, Task P3) running on a processor.

These tasks are defined by their execution time (called capacity) (C) their period (T) and their relative deadlines (D):

Task P1: C1 = 2, T1 = 5, D1 = 5
Task P2: C2 = 2, T2 = 7, D2 = 7,
Task P3: C3 = 5, T3 = 30, D3 = 30

We assume that priorities are assigned according to “rate monotonic” and a Highest Priority First scheduling method.

a) What is the maximum response time of P1, P2 and P3?

Now we assume the same task set and that tasks P1 and P3 share a common resource via a semaphore S. Task P3 uses S for 1 time unit and P3 uses S for 1 time unit. Priority Inheritance Protocol (PIP) is used to control the access to S.

Name	Execution	Release instant
P1	+S-E	2
P2	EE	1
P3	+S-EEEE	0

b) What are the blocking factors (B1, B2, and B3) of tasks P1, P2 and P3?
c) What is the maximum response time of P1, P2 and P3?

IV. Scheduling in Real-Time Operating Systems (5 points)

The following text is taken from *Wikipedia*. Do read it carefully, and answer the following questions:

(a) Why do tasks spend most of their time in "blocked" state? How can a task block? How can it unblock?

(b) Why should preemption be allowed when searching for the next task to execute?

(c) Could you explain why a linked list is not adequate when having both non real-time and real-time tasks in a system? What data structure would you propose to use? Justify your answer!

Excerpt from: http://en.wikipedia.org/wiki/Real-time_operating_system:

Scheduling

In typical designs, a task has three states: 1.running 2.ready 3.blocked. Most tasks are blocked, most of the time. Only one task per CPU is running. In simpler systems, the ready list is usually short, two or three tasks at most.

The real trick is designing the scheduler. Usually the data structure of the ready list in the scheduler is designed so as to minimize the worst-case length of time spent in the scheduler's critical section (during which preemption is inhibited, and, in some cases, all interrupts are disabled). But the choice of data structure depends also on the maximum number of tasks that can be on the ready list.

If there are never more than a few tasks on the ready list, then a simple unsorted bidirectional linked list of ready tasks is likely optimal. If the ready list usually contains only a few tasks but occasionally contains more, then the list should be sorted by priority, so that finding the highest priority task to run does not require iterating through the entire list. Inserting a task then requires walking the ready list until reaching either the end of the list, or a task of lower priority than that of the task being inserted. Care must be taken not to inhibit preemption during this entire search; the otherwise-long critical section should probably be divided into small pieces, so that if, during the insertion of a low priority task, an interrupt occurs that makes a high priority task ready, that high priority task can be inserted and run immediately (before the low priority task is inserted).

The critical response time, sometimes called the ***flyback time****, is the time it takes to queue a new ready task and restore the state of the highest priority task. In a well-designed RTOS, readying a new task will take 3-20 instructions per ready queue entry, and restoration of the highest-priority ready task will take 5-30 instructions. On a 20MHz*

68000 processor, task switch times run about 20 microseconds with two tasks ready. 100 MHz ARM CPUs switch in a few microseconds.

In more advanced real-time systems, real-time tasks share computing resources with many non-real-time tasks, and the ready list can be arbitrarily long. In such systems, a scheduler ready list implemented as a linked list would be inadequate.