

# Examen de Système

14 janvier 2010

Duration: 3 hours. All course materials can be used. Answers can be given in either English or French. Justify all your answers.

1. What are the additional tasks of a multi-tasking, multi-user system, as opposed to a single-tasking, single-user system? Justify why these additional tasks are necessary. Which additional hardware features are necessary to create a multi-tasking, multi-user system?
2. In the Unix shell, one can “pipe” output from one command to another, e.g. `cmd1 | cmd2`. Consider the following, alternative sequence: `cmd1 > temp; cmd2 < temp; rm temp`. Does the second sequence work whenever the first sequence works? When it does work, will it produce the same output and finish in the same amount of time?
3. Give shell commands that accomplish the following tasks. Note: In the following, “text file” means a file with the ending `.txt`.
  - (a) Exchange the contents of two files `titi` and `toto`.
  - (b) Give the number of text files in the current directory.
  - (c) Copy the contents of all text files into a file called `toto`.
  - (d) Suppose that the file `titi` contains a list of file names. Copy the contents of all files in that list into a file called `toto`.
4. Write some C functions that accomplish the following tasks. You may use any existing system calls or C library functions. (Minor syntax errors will be tolerated, as long as the meaning of the code remains clear. Ditto for the order or number of arguments in system calls.)
  - (a) Given two strings `s` and `t`, find whether `s` is a prefix of `t`; return 1 if that’s the case, 0 otherwise. (E.g., the function should yield 1 for `dang` and `danger`.)
  - (b) Given two strings `s` and `t`, treat them as the names of two files and compare their contents. Return 1 if the contents differ and 0 otherwise. To simplify the task, no error handling is required, you may assume that all operations on files succeed.

5. Suppose that, on a multi-tasking system, a user process issues a system call to read a chunk of data from a file stored on the hard drive. Describe the interaction between process, operating system, and hardware that takes place until the system call is finished, assuming that Direct Memory Access (DMA) is used.
6. We consider an algorithm for mutual exclusion between  $n$  processes, numbered  $0 \dots n - 1$ , which is based on message passing.

Messages transfer is reliable, i.e. no message gets lost, no message is duplicated or modified, and messages cannot overtake one another, i.e. messages between any two processes are guaranteed to arrive in the order in which they were sent. However, a message may take an arbitrary finite time to be delivered.

Messages are of two types, simply *token*, or *sendto(i)*, where  $i$  is a process number. A process is either *idle*, *waiting*, *critical*, or *active*. Initially, process 0 is active, all others are idle. Each process has a variable *fwd*, which is initially 0 for all processes.

All processes behave as follows:

- An idle process  $i$  may decide that it wants to enter the critical section. It then sends a message *sendto(i)* to the process indicated by its variable *fwd* and becomes waiting.
- If a waiting process receives the *token* message, it becomes critical for some finite amount of time, after that it is active.
- If an active process receives the message *sendto(i)*, it will send the message *token* to process  $i$ , set  $fwd := i$ , and become idle.
- If an idle or waiting process receives the message *sendto(i)*, it will send that message on to the process indicated by its variable *fwd*.

Questions:

- (a) Consider the graph whose nodes are  $0 \dots n - 1$ , and we have an edge from  $j$  to  $i$  if the *fwd* value of process  $i$  is  $j$ , and  $i$  is either idle or waiting, and no *token* message is currently on the way to  $i$ . Show that, at any time during the algorithm, this graph is in fact a tree.
- (b) Is mutual exclusion guaranteed, i.e. is there at most one critical process at one time?
- (c) Are there deadlocks in the protocol, i.e. could we reach a state in which no process can become critical anymore?
- (d) Is the protocol fair, i.e. will every waiting process become critical after a finite amount of time? (Assuming that every process can make computation steps every now and then, as usual.)

7. Consider processes  $0 \dots n - 1$  which are arranged in a ring, i.e. each process  $i$  sends data only to process  $(i + 1) \bmod n$ . We call  $i$  the *feed* of  $(i + 1) \bmod n$  and  $(i + 1) \bmod n$  the *recipient* of  $i$ . We make the same assumptions about reliability etc. of messages as in the previous question, except that we use different types of messages:  $id(s)$ , where  $s$  is an identifier, and  $winner(j)$ , where  $0 \leq j < n$ .

Each process  $i$  has some number  $s_i$ , which serves as a unique identifier, i.e.  $s_i = s_j$  if and only if  $i = j$ . A process is either active or inactive.

We propose the following algorithm as a leader-election protocol by describing the behaviour of process  $i$ , where  $0 \leq i < n$ :

Initially, each process  $i$  is active. Its first action is to send the message  $id(s_i)$  to its recipient. Then it waits for messages from its feed and treats them as follows:

If the received message is  $id(s)$ , then:

- if  $s < s_i$ , the message is ignored;
- if  $s > s_i$ , the process becomes inactive and repeats the received message to its recipient;
- if  $s = s_i$ , the process sends the message  $winner(i)$  to its recipient.

If the received message is  $winner(j)$ , then:

- if  $i = j$ , the message is ignored;
- if  $i \neq j$ , the process repeats the received message to its recipient.

Questions:

- (a) Is the protocol correct, i.e. will it always terminate in a state where exactly one process is still active, and only that process has sent a winner message, and no more messages are waiting to be received?
- (b) For the cases where the protocol works correctly, what is the maximum number of messages exchanged in this protocol, as a function of  $n$ ? Compare with the protocol by Dolev et al.