“Operating Systems” Course

Final Examination – Fall 2008

February, 2009

Duration: 2h

ludovic.apvrille@telecom-paristech.fr

Documents regarding Operating Systems (OS) or RTOS are not 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 that were given to you during the two following lecture sessions: January 22th and February 1st. Having other documents (or communicating devices) with you shall be considered no less than 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 both!

I. Understanding of the course (7 points)

(a) What is the difference between the supervisor mode of a microprocessor and the administrator / root rights provided by an operating system? Are those two modes related? If yes, closely explain in what they are related. (2 points)

(b) Explain in 5 lines at most the role of a driver. Then, explain why drivers frequently need to rely on buffers for managing devices. At last, explain why, when removing a USB key from a computer running Linux (or Solaris, Windows, etc.), one must first “detach the device”. (3 points)

(c) What are preemptions points used for? What happens if they are put too frequently in the operating system? On the contrary, what happens if they are not frequently put in the operating system? Why are preemption points more particularly at stake in real-time operating systems? (2 points)

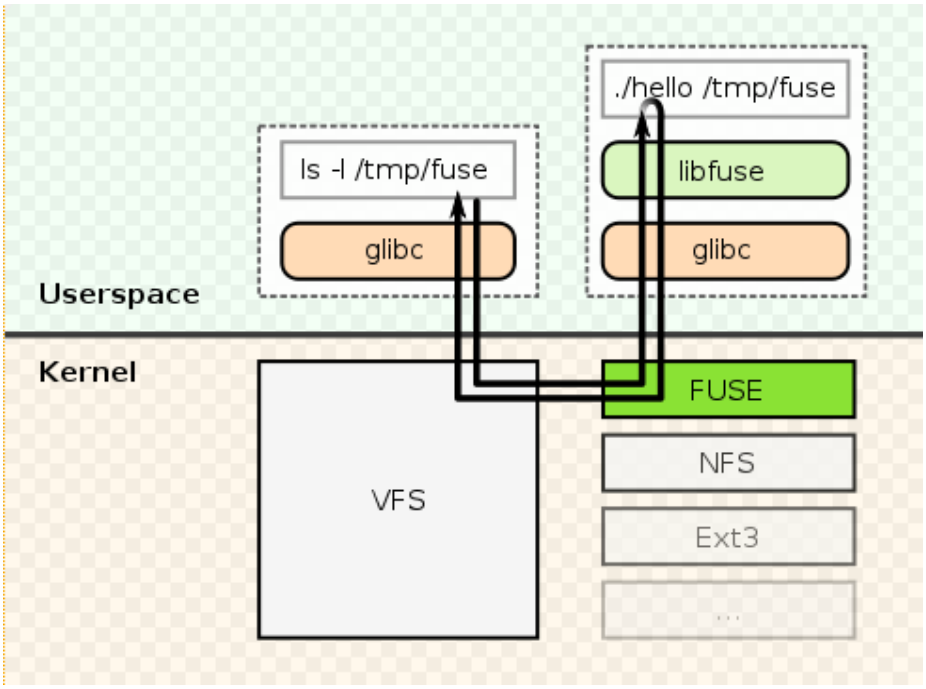
II. File systems in user space (7 points)

The following text and figure are taken from Wikipedia.
Filesystem in Userspace (FUSE) is a loadable kernel module for Unix-like computer operating systems, that allows non-privileged users to create their own file systems without editing the kernel code. This is achieved by running the file system code in user space, while the FUSE module only provides a "bridge" to the actual kernel interfaces. FUSE was officially merged into the mainstream Linux kernel tree in kernel version 2.6.14.

FUSE is particularly useful for writing virtual file systems. Unlike traditional filesystems, which essentially save data to and retrieve data from disk, virtual filesystems do not actually store data themselves. They act as a view or translation of an existing filesystem or storage device. In principle, any resource available to FUSE implementation can be exported as a file system. See Examples for some of the possible applications.

Released under the terms of the GNU General Public License and the GNU Lesser General Public License, FUSE is free software. The FUSE system was originally part of A Virtual Filesystem (AVFS), but has since split off into its own project on SourceForge.net.

FUSE is available for Linux, FreeBSD, NetBSD (as PUFFS), OpenSolaris and Mac OS X.



You may consider, in the following questions, that a file system is a tree whose nodes are either files or directories. A file cannot have a subtree. On the contrary, a directory node has a subtree.

(a)Recall what is the *Virtual File System*, and what are the main system calls to use it. (1 point)

(b)Explain, when you want to implement a new file system based on FUSE, what has to be implemented in the kernel, and what has to be implemented in the userspace? (1 point).

The following text has been taken from Linux Gazette, January 2007:

curlftpfs: Mount FTP servers

This is something that I really love! Accessing a FTP server as though it's contents were on directories on your own computer! Just get `curlftpfs` *from the curlftpfs page, install it using the standard* `./configure; make; make install`*, or install the package available for you distribution, and just do something like this:*

```
[kumar@debian ~] mkdir IITM_Mirror
[kumar@debian ~] curlftpfs ftp.iitm.ac.in IITM_Mirror/
[kumar@debian ~] cd IITM_Mirror/
[kumar@debian ~/IITM_Mirror] ls
...
README          debian...
```

That's it! I have used `IITM_Mirror` *as the mount point for the mirror. You can now mount FTP servers, even with password login, so that you can do uploads as well. Do* `curlftpfs -h` *for learning how to mount servers with login for write access and using proxies. To unmount, use* `fusermount -u ~/IITM_Mirror`*.*

In the next questions, we consider that ftp offers the following commands:

- **open**, to open a connexion with a remote computer [the robot]
- **quit**, to close the connexion with the remote computer
- **ls** to list remote files
- **put**, to put a file on the remote filesystem
- **get**, to get a file from the remote file system to the local filesystem
- **mkdir** to create a directory
- **rmdir** to remove an empty directory
- **cd** to change of directory on the remote filesystem

Also, we consider that a user may do the following six system calls on the virtual file system: **ls**, **write**, **read**, **mkdir**, **rmdir**, **cd**.

(c)Explain what probably happens in the operating system and in the curlftpfs application for each commands of the example (mkdir, curlftpfs, etc.). (1 point).

(d)For the 6 system calls commands listed above, explain how they could be implemented in C at the FUSE application level (i.e. in *culrftpfs*). I do not ask you to provide the full code, but rather a sketch of what could be done. You may obviously, in your C files, make calls to ftp commands (get, put, etc.). Also, in your implementation, do take into account the fact that several commands may be sent at the same time to the *curlftpfs* FUSE application (explain how you handle this). Also, for commands with similar implementation, you may just provide only one implementation and precise a list of commands to which this implementation applies. At last, to propose an implementation, do assumptions on interfaces provided by *libfuse* (4 points).

III. Scheduling tasks (5 points)

Question 1:

Consider a set of 3 independent tasks (Task P1, Task P2, Task P3) running on a processor.
These tasks are defined by their periods (T) their worst case execution times (called capacity) (C)
P1: T1=4, C1=1
P2: T2=6, C2=2
P3: T3=10, C3 = 3
Priorities are assigned according to Rate Monotonic.

(a) Compute the Hyperbolic bound of this task set. Can you conclude by this method on the schedulability?
(b) Verify the schedulability of this task set

Question 2:

Consider the following task set:
Task P1 : T1 = 10, C1 = 4,
Task P2 : T2 = 15, C2 = 3,
Task P3 : T3 = 20, C3 = 4,

P1, P2 and P3 are using the same resource S in the following manner:
P1 : +S- (1 unit of time using S)
P2 : +SSSSS- (5 units of time using S)
P3: +SSS- (3 units of time using S)

We assume S is taken according to PIP (Priority Inheritance Protocol) rules

(a) Compute the blocking factor Bi of each task
(b) Verify the schedulability of this task set under Rate Monotonic