Exam
Operating Systems - OS
Solution

Ludovic Apvrille
`ludovic.apvrille@telecom-paristech.fr`

February, 2016

Authorized documents: Nothing! The grading takes into account the fact that you don't have any documents 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.
Solutions are provided within this font/color style. You may naturally get the maximum grade with a different solution. Also, please contact me if you find any improvement to this solution.

1 Course understanding (7 points, ∼30 minutes)

a. What is the interest of double buffering for drivers? Provide an example to illustrate your answer. [3 points]
Double buffering is a software technique - commonly used by drivers - in order to perform simultaneous read/write operations on a data flow. Thus, a producer first writes data to buffer #1. When buffer 1 is full, then the buffer is "given" to the consumer, and the producer can continue writing data in another buffer (buffer #2), and so on. This technique is useful when there is a difference in bandwidth between a producing device and a reading application. For example, a modem is commonly much slower than the protocol stacks of the Operating systems,so double buffers are used in the drivers of modems.

b. A process may contain several threads. What are the memory areas which are shared by the threads of one given process, and which areas belong only to one thread? Why is it so? Provide a pseudo-code that illustrates the sharing of a memory area - e.g. a variable - between two processes, and between two threads of the same process. [4 points]

All the threads of one given process share the whole address space of the process apart from the stack, that is, the heap, the code (text section), the global variables and the environment data. There is one stack reserved for each thread. The stack of a thread is used to store the function calls of that given thread. Yet, nothing prevents a thread from writing in the stack of another thread. Said differently, the MMU or the operating system cannot detect that a thread is writing in the stack of another thread.

To share a variable between two processes, here is how it can be done. Let's start with process #1.

(a) Create a shared memory segment.

(b) Map this shared memory segment into the address space of the current process.

(c) Allocate a variable in that mapped area, and save the address.

In process #2:

(a) Open the shared memory segment created by process #1

(b) Map this shared memory segment into the address space of the current process.

(c) Read the value at the address of the variable in the shared address space.

With threads, things are much easier, because a process global variable is intrinsically shared between two threads of this process.

(a) int sharedVariable ...

(b) main()

- *Create thread1 and thread2 // Thread1 and thread2 can use sharedVariable*

2 Linux Ram disk (11 points, ∼90 minutes)

File partitions in Linux can also be set directly in RAM. Below, you will find an article discussing two different RAM-based file systems (ramfs, tmpfs). Carefully read this article, and answer to the following questions.

a. Provide a summary of this article in 150 words (+/- 10 words). [4 points]

(This summary is the one of Federico Madotto).
Modern Linux distributions permit the creation of a particular type of storage area very efficient in terms of R/W performance. This is useful for applications that intensively use cached data. This mechanisms is based on the allocation of a portion of the RAM which can be mounted on the file systems as if it were a "normal" folder. There are two different types of RAM-based file systems that implement this mechanism: ramfs and tmpfs. ramfs - the older - uses the same mechanism that Linux uses for caching recently accessed files, with the exception of not being

flushed when the used memory exceeds a threshold. Another drawback is that the size of the file system cannot be precisely estimated. A size limit can be specified with tmpfs, resulting in a more manageable file system, but since it may use swap space, files may have to be read from the disk, reducing the overall performance.

b. Comment on the current memory/swap state of the Linux machine of the author. [2 points]
Most of the RAM is allocated (29GB out of 31GB). Out of these 29GB of allocated RAM, about 8GB are used by buffers/cache. The actual amount of RAM used by the system and applications is thus 20GB. As buffers and cache could be removed (at the cost of performance), this leaves 11GB actually usable. Also, 6GB out of the 13GB of the swap partition are used, which means that the amount of memory used by the applications and buffers/cache would be 35GB and so the system must either swap part of its memory (this is the case here) or reduce the amount of memory used by buffers and cache.

c. Could you be more precise on what the author meant by "As the data is lost when the machine reboots the data must not be precious as even scheduling backups cannot guarantee that all the data will be replicated in the event of a system crash" [2 points]
The author means that the partition is volatile i.e. whenever the system reboots or crashes, data stored on the partition is lost. So, only data that has been copied to a non volatile memory before the crash - e.g., during a backup - can be recovered.

d. I have done the following test on my Linux machine: I have compiled the latex sources of the RTOS slides from the same state (i.e., after removing all intermediate files of the compilation process) in two cases: (i) with the files located on my SSD, and (ii) with the same files located in a Ramdisk. **Comment on the results** with regards to what is stated in the article. **Imagine a better test** to evaluate the difference between an SSD and a ramfs. [4 points]
There is a significant improvement when using the tmpfs with regards to the SSD. This difference mostly applies to the overall time, and is not really significant for the process or the OS processing time. It probably means that most of the extra time with the SSD is due to I/O blocking delays. Yet, the author of the article was claiming an improvement of a factor of 10, and the test demonstrates a much lower improvement. This is probably due to our test not doing only file accesses, but also intensive computations - compilation - for which the use of tmpfs has no impact.
A better test would thus either evaluate the two file systems for file operations only - reading/writing large files, small ones,, a mix of the two, etc. -, or on the contrary, performing an evaluation for more realistic applications, e.g. with video games, office applications, multimedia, etc.

SSD:

```
$ make ultraclean&&time make all
real    0m34.464s
user    0m22.256s
```

```
sys     0m6.528s
```

Ramdisk (tmpfs): I have first created the ramdisk, then, I have mounted it in /mnt/ramdisk. Finally, I have copied the files from the SSD and compiled the latex sources:

```
$ sudo mkdir /mnt/ramdisk
$ sudo mount -t tmpfs -o size=1024m tmpfs /mnt/ramdisk
$ cd /mnt/ramdisk
$ cp -R /homes/apvrille/slidesRTOS .
$ make ultraclean&&time make all
real    0m26.788s
user    0m21.192s
sys     0m6.052s
```

The Difference Between a tmpfs and ramfs RAM Disk

Dec 2013.
Taken from http://www.jamescoyle.net/knowledge/951-the-difference-between-a-tmpfs-and-ramfs-ram-disk

There are two file system types built into most modern Linux distributions which allow you to create a RAM based storage area which can be mounted and used like a normal folder.

Before using this type of file system you must understand the benefits and problems of memory file system in general, as well as the two different types. The two types of RAM disk file systems are tmpfs and ramfs and each type has it's own strengths and weaknesses.

What is a memory based file system (RAM disk)?

A memory based file system is something which creates a storage area directly in a computers RAM as if it were a partition on a disk drive. As RAM is a volatile type of memory which means when the system is restarted or crashes the file system is lost along with all it's data.

The major benefit to memory based file systems is that they are very fast – 10s of times faster than modern SSDs. Read and write performance is massively increased for all workload types. These types of fast storage areas are ideally suited for applications which need repetitively small data areas for caching or using as temporary space. As the data is lost when the machine reboots the data must not be precious as even scheduling backups cannot guarantee that all the data will be replicated in the event of a system crash.

tmpfs vs. ramfs

The two main RAM based file system types in Linux are tmpfs and ramfs. ramfs is the older file system type and is largely replaced in most scenarios by tmpfs.

ramfs creates an in memory file system which uses the same mechanism and storage space as Linux file system cache. Running the command free in Linux will show you the amount of RAM you have on your system, including the amount of file system cache in use. The below is an example of a 31GB of ram in a production server.

```
$ free -g
                    total  used  free  shared  buffers  cached
Mem:                 31     29     2      0       0       8
-/+ buffers/cache:          20     11
Swap:                13     6      7
```

Note: free displays the total amount of free and used physical and swap memory in the system, as well as the buffers used by the kernel. The "-g" option is used to display the amount of memory in gigabytes.

Currently 8GB of file system cache is in use on the system. This memory is generally used by Linux to cache recently accessed files so that the next time they are requested then can be fetched from RAM very quickly. ramfs uses this same memory and exactly the same mechanism which causes Linux to cache files with the exception that it is not removed when the memory used exceeds threshold set by the system.

ramfs file systems cannot be limited in size like a disk base file system which is limited by its capacity. ramfs will continue using memory storage until the system runs out of RAM and likely crashes or becomes unresponsive. This is a problem if the application writing to the file system cannot be limited in total size. Another issue is you cannot see the size of the file system in df and it can only be estimated by looking at the cached entry in free. tmpfs

tmpfs is a more recent RAM file system which overcomes many of the drawbacks with ramfs. You can specify a size limit in tmpfs which will give a 'disk full' error when the limit is reached. This behaviour is exactly the same as a partition of a physical disk.

The size and used amount of space on a tmpfs partition is also displayed in df. The below example shows an empty 512MB RAM disk.

```
$ df -h /mnt/ramdisk
Filesystem Size     Used    Avail      Use% Mounted on
tmpfs       512M      0      512M       0%   /mnt/ramdisk
```

These two differences between ramfs and tmpfs make tmpfs much more manageable however this is one major drawback; tmpfs may use SWAP space. If your system runs out of physical RAM, files in your tmpfs partitions may be written to disk based SWAP partitions and will have to be read from disk when the file is next accessed. In some environments this can be seen as a benefit as you are less likely to get out of memory exceptions as you could with ramfs because more 'memory' is available to use.