INFO0940 - Operating systems

Project report

Step number 5

Romain Compagnie Olivier Moitroux
Group 19
Third year engineering bachelor

19 avril 2018

The goal of this report is to discuss the main differences between using the read/write system calls and the mmap system call to read or write in a file.

1 Read/Write

The read/write system calls require a valid file descriptor and an allocated buffer in process space to perform I/O operations. The size of the buffer will affect the file operation properties.

A call to the read system call usually leads to the following steps being executed.

- The kernel searches the page cache for the requested memory.
- If the page cache does not contain the requested memory, a sufficient number of page frames are allocated and the requested data is copied from disk to these page frames.
- The kernel copies the requested data from the page frames to the user space buffer.

A call to the write system call usually leads to the following steps being executed.

- The kernel searches the page cache for the memory to be modified.
- If the page cache does not contain the requested memory, a sufficient number of page frames are allocated and the requested data is copied from disk to these page frames.
- The kernel copies the content of the user space buffer containing the data to be written to the corresponding page frames.
- The writing is reflected to disk when the file buffers are flushed.

1.a Advantages

- No setup operations required.

1.b Disadvantages

- The data of the I/O operation is present in a user buffer and in the kernel page cache in physical memory, wasting memory.
- CPU time is needed to copy data between the user buffer and the kernel page cache.

1.c Efficient usage

The read/write system calls are usually faster when reading small files in a sequential fashion.

2 Mmap

The reading and writing operations using memory-mapped files can be set with the mmap system call. This system call requires a valid file descriptor.

The mmap system call allows to directly map pages in main memory to the virtual address space of the process. This allows direct access to data in file pages, without a user space buffer. The pages are however not directly copied to main memory.

An operation on memory-mapped files triggers a page fault if the data accessed is not present in a page frame in main memory. The kernel then handles the loading of the page containing the requested data from disk to physical memory.

2.a Advantages

- No process buffer in user space needed.
- Aside from any potential page faults, mmap does not require any system call or context switch overhead to read or write data.
- Can be used for inter-process communication by mapping the same file in different processes.
- Many processes requiring the same memory can all map the same pages containing the memory, avoiding wasted memory.
- Memory mapping allows the programmer to keep using pages from the cache until the job is done (if there are a lot of opening/closing f.e.). With read/write, the cache may have been flushed meanwhile.

2.b Disadvantages

- Changing virtual memory mapping is a costly operation as it has important overhead.
- The process address space must contain a contiguous available block of addresses to map the file present in the page cache. This can be an issue when operating with large files on a 32-bit machine.
- The mapping must start on a multiple of the page size.
- The mapped area is always a multiple of the page size.
- mmap can't be used on things like pipes and ttys ; read / write are the only way you can work with some types of files.

2.c Efficient usage

Memory-mapping is recommended if large files are accessed randomly or in an unpredictable fashion. It also allows sharing the file with other processes.

3 Quantitative analysis

Average execution times of the two methods have been measured for a sequential reading on a file, a sequential writing, a random reading and a random writing, for different file sizes. For every test, operations on a number of bytes equivalent to the size of the entire file have been performed. The execution times were then averaged, based upon 10 measurements. These times do not include the file opening and closing. For mmap, the mapping time is included.

The user space buffer required by the read/write system calls used for sequential tests had a size of 16 kB, which is a reasonable size to use when the sizes of the reading or writing operations is not known in advance. When operating on randomly located bytes, a buffer with a 1 byte size was used.

As can be seen on Figures 1 and 2, the execution times of reading and writing operations on files with sizes ranging from 1 kB to 160 MB are similar. The method using mmap seems to have a small advantage when operating on small files.

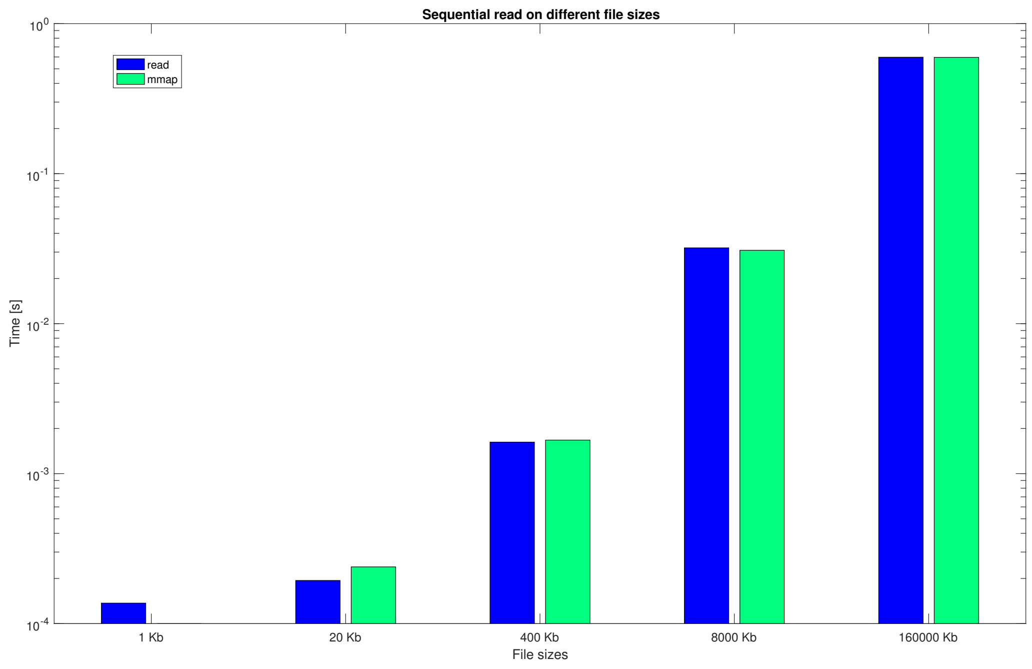

FIGURE 1 – Average sequential reading times using write and mmap, on a logarithmic scale

However, we can see on Figures 3 and 4 that the execution times are completely different when operating randomly on bytes. The graphs in these figures have a logarithmic scale. In this case, in reading as well as in writing, mmap is much faster than the read/write system calls. Indeed, the read/write system call require a system call with much overhead for each byte operation, while mmap only requires accessing the mapped memory through pointer arithmetic. For reading and writing operations, mmap is about 100 times faster when operating on file sizes of 625 kB.

Figure 5 shows the distribution between user and system CPU time for sequential and random reading for the read system call and mmap, on a logarithmic scale. On sequential reading, the total time spent is similar for both methods. However, read require more time spent in system mode. On random reading, we can see that read requires much more time to execute, because of the huge amount of CPU cycles required in system mode to copy the contents of the file to the user process buffer.

4 Conclusion

In conclusion, we can say that mmap is much faster than the read/write system calls when the access pattern is not sequential. In sequential access, the comparison does not show a clear advantage of one method over the other.

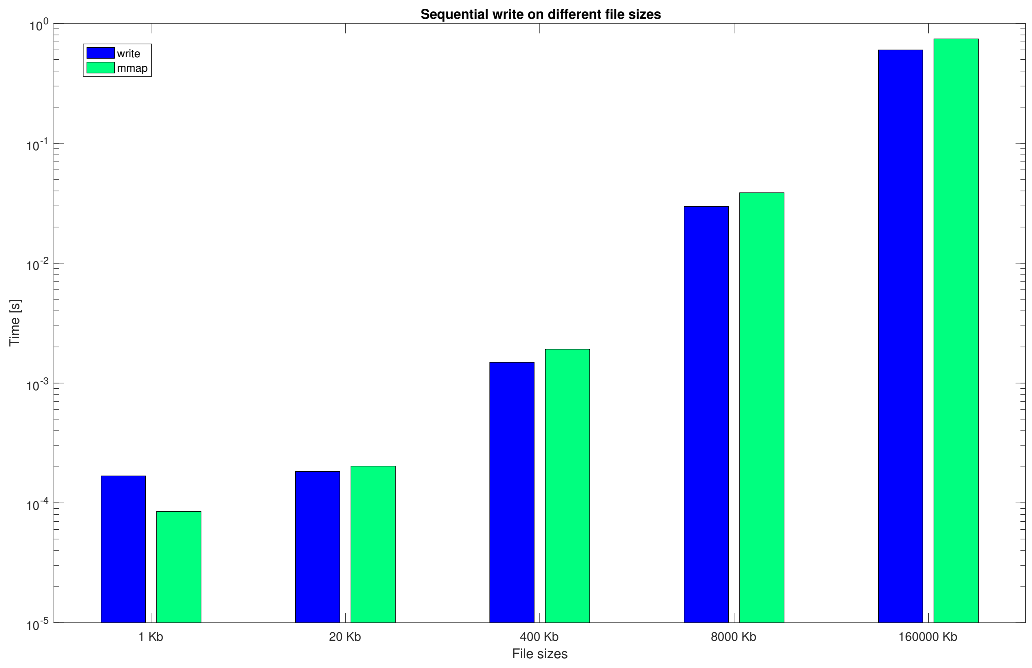

FIGURE 2 – Average sequential writing times using write and mmap, on a logarithmic scale

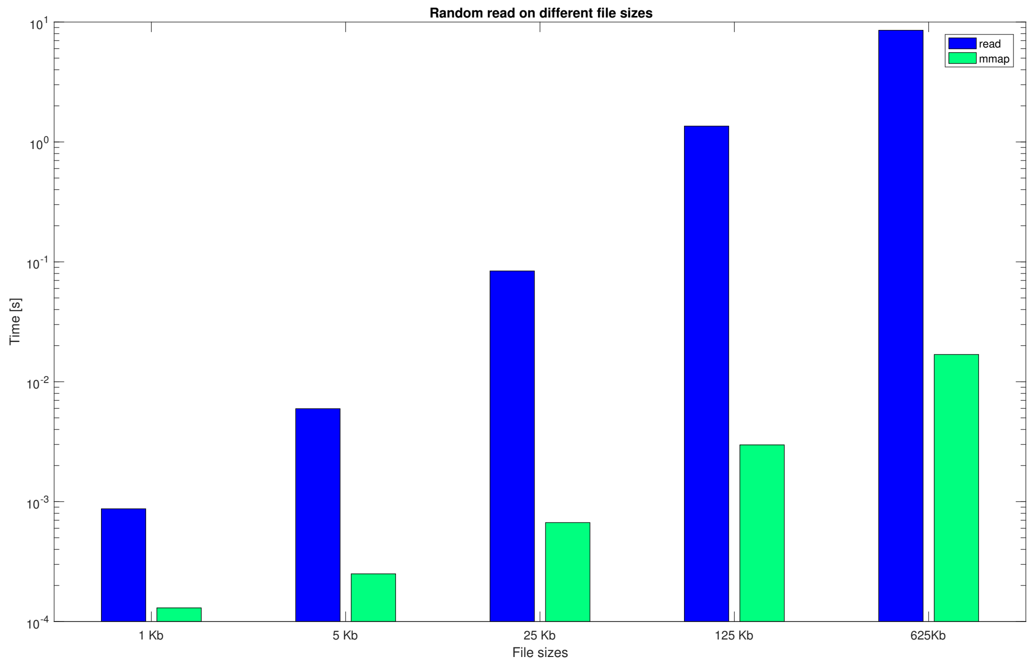

FIGURE 3 – Average random reading times using write and mmap, on a logarithmic scale

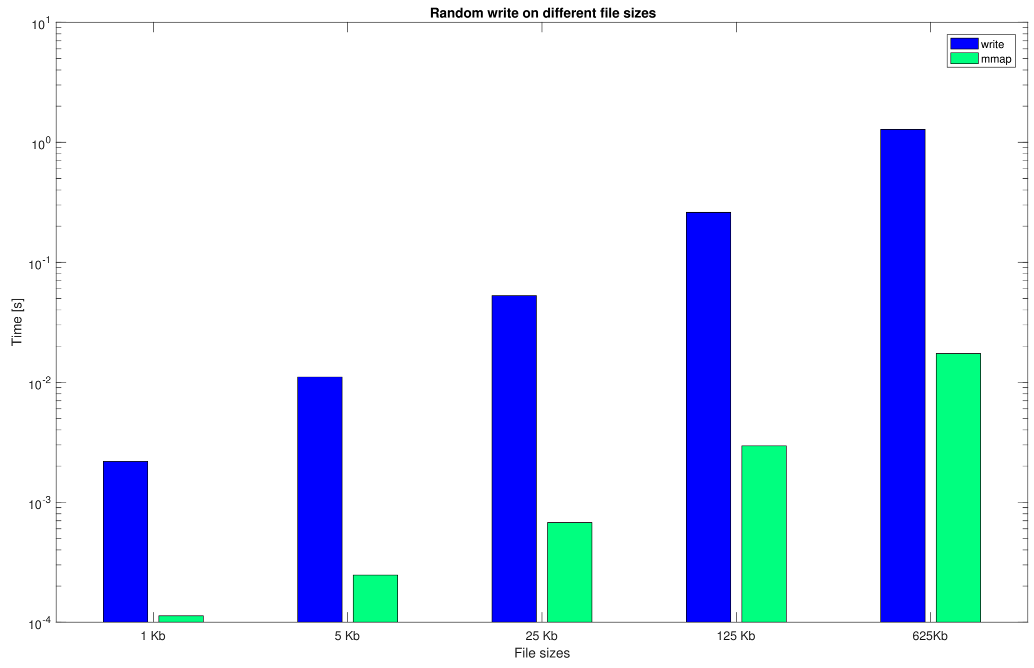

FIGURE 4 – Average random writing times using write and mmap, on a logarithmic scale

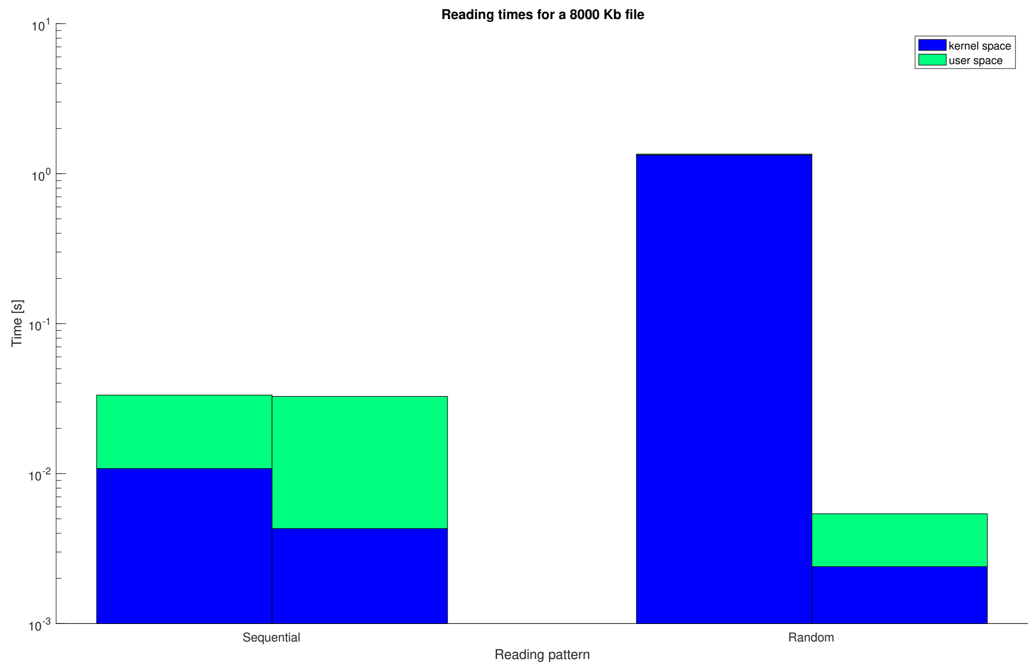

FIGURE 5 – Distribution between user and system CPU time for sequential and random position reading on a 8000 kB file, using read (on the left parts of the bars) and mmap (on the right parts of the bars), on a logarithmic scale