Lab 04 - Martino Mensio

Exercise 1

The first exercise implementation is straightforward, doing little modifications from lab1.1. The program simply generates n (command line parameter) integers randomly and stores them in a file in binary representation.

Exercise 2

The second exercise is designed to sorting the array represented in the file using a memory-mapping to a file. The initial part of the program, after parsing the command line parameter (the name of the file), tries to open the file and get some information about it: the size of the file, needed to understand how many integer numbers are stored.

Then the mapping to the memory is done via the system call **mmap**, using the flag MAP_SHARED in order to make the results visible to other processes and to allow the operating system to sync with the file system (and therefore having the file sorted after the end of the execution). If the MAP_PRIVATE was used, the operating system would probably have applied the Copy-On-Write mechanism, so the modifications to the file could not be persistent.

The sorting procedure works in this way:

1 – A specific number of threads is created, and each one is assigned a different slice of the array

2 – Each thread sorts its slice of the array

3 – The last thread that terminates his local sort is required to do extra job:

- Check for every boundary the values on the left and on the right
- If some couple of values don't follow the sorting order
- Sets the number of swaps occurred

4 – Each thread checks the number of swaps occurred. If some swaps have been done, they re-order the slice assigned (going to point 2).

Each thread is provided with a parameter (of type mergeParamType) that contains a pointer to the array and the extreme indexes of the region assigned.

In order to accomplish this behavior, some global variables and structures have been created and are visible to all of the threads:

- A counter for the number of swaps occurred in the last iteration
- A counter for the number of threads that are currently sorting their slice
- A mutex to protect the counter of actively sorting threads
- An array of the parameters provided to the pool of threads, so that the last thread can locate the boundaries and perform the necessary swaps operations
- Two semaphores to implement a double barrier mechanism to guarantee that the threads execute in the desired order

The global counter of active sorting threads (runningThreadsSorting) is accessed in exclusive way (not concurrently) by the threads surrounding the reads and writes to the variable with a lock and an unlock on the mutex.

Instead the number of swaps (swapsCount) is protected thanks to the fact that the writes are done only by the last thread, and the reads are done in the while check after the two barriers, so there is no problem accessing this variable. The last thread is writing when all the other threads are waiting on semaphore1 or in a near instruction (surely after the decrease of runningThreadsSorting and the release of the mutex).

The two barriers are needed because the processes are cyclic. In this way race conditions are not possible.