Report on exercise #3

Matteo Corain S256654

Laboratory #3 – System and device programming – A.Y. 2018-19

The proposed solution makes use of a global integer `size` threshold and it defines a quicksort data structure (`struct qs_data_s`) to be used for passing data to the threaded quicksort calls; this structure includes:

- A pointer to the vector of integers to sort (`*v`);
- A `left` and a `right` integer bounds to delimit the portion of the vector to be sorted.

The main function performs the following actions:

- It checks the number of input arguments (via the `argc` variable), then sets the value of `size` by calling `atoi()` on the first argument;
- It opens the input file, whose name is passed as the second program argument, in read-write mode by using the `open()` system call with the `O_RDWR` flag, checking the correctness of the operation;
- It maps to memory the opened file descriptor, checking the correctness of the operation, and stores the memory address of the obtained mapping into the variable `v` (of type `int*`); for this purpose, it uses the `mmap()` system call to which the following arguments are passed:
  - `NULL` as the hint for memory placement of the file;
  - `SIZE * sizeof(int)` as the size of the file to map (`SIZE` integer, set to 100, numbers are stored in the mapped file);
  - `PROT_READ | PROT_WRITE` as the required memory protection (we need access both for reading and for writing);
  - `MAP_SHARED` as the access flag (so that changes performed in memory are propagated to the file on the disk);
  - `0` as the offset (we map the file from the beginning).
- It prepares a structure of type `struct qs_data_s` and passes that to `quickthread_wrapper()`, by setting the pointer to the location of the memory-mapped file, the left bound to `0` and the right bound to `SIZE` (all the array should be sorted); then, based on the return value of the function, it possibly joins the created thread;
- It prints the sorted vector;
- It finally unmaps (via `munmap()`) and closes (via `close()`) the input file.

The `quicksort_wrapper()` is used to wrap the call to `quicksort()` in a way that it is automatically managed the choice between performing the call in the current or in a new thread. In particular, the function, which takes as arguments a pointer to a `struct qs_data_s` variable and a pointer to a `pthread_t` variable, performs the following actions:

- It checks whether the difference between the right and the left bounds of the array to be sorted is less than the configured `size` threshold and, in that case, it simply calls `quicksort()` on the given `struct qs_data_s` variable, returning `QUICK_REC` (mapped to `0`);
- Otherwise, it creates a new thread to run the `quicksort()` function on the given `struct qs_data_s*` variable, storing the thread identifier in the `pthread_t*` variable and returning `QUICK_THR` (mapped to `1`).

In both cases, an informational message is printed. The `quicksort()` function is a slightly modified version of the standard quicksort algorithm, with the following changes:

- It takes a single argument of type `struct qs_data_s*`, from which the vector to be sorted and its left and right bounds are obtained;
- It prepares two structures of type `struct qs_data_s`, to be passed to the recursive or threaded calls to `quicksort()` working on the left and right parts of the vector; in particular, the pointer is always set to the location of the memory-mapped file, while the left and right bounds are set to `left` and `j` in the case of the left portion and to `j+1` and `right` in the case of the right portion;
- It calls twice the `quicksort_wrapper()` function, once on the left side of the vector and once on the right side of the vector;
- It checks if the call to `quicksort_wrapper()` resulted in a thread activation and, if so, it joins the created thread before terminating.