Report on exercise #1

Matteo Corain S256654

Laboratory #2 – System and device programming – A.Y. 2018-19

The proposed solution makes use of the integer global variables g (used to store the values read by the client threads and modified by the server thread) and `thread_complete[2]` (used as flags to indicate the termination status of the two client threads), plus three global semaphores `sem_server`, `sem_client` and `sem_print`. As far as data structures are concerned, the structure `thread_data_s` (used to pass values to the thread functions) has been defined, storing an integer identifier for the client thread and the name of the file to process.

The main thread allocates (via a `malloc()` operation) and initializes (via a `sem_init()` operation) the three semaphores, each one having `0` as initial value, checking the correctness of the different operations. Then, it sets up the array of two thread data structures in the following way:

- ID set to 0 and filename set to `fv1.b` for the first client thread;
- ID set to 1 and filename set to `fv2.b` for the second client thread.

After that operation, the main thread creates the two client threads by means of a call to `pthread_create()` (checking the correctness of the operation), using as thread function `thread_runner()` and passing the address of one of the two thread structures as parameter. Finally, it enters in a `while` loop which terminates when both of the client threads have set their termination flag, signals on `sem_client` to unlock one of the two clients and starts waiting on `sem_server`.

Each of the client threads retrieve their parameters from the passed data structure, open the respective binary file in read mode (checking the correctness of the operation) and read an integer value on variable `next`, before entering in a loop until the end of the file is reached and starting to wait on `sem_client`. When the server thread posts on `sem_client`, one of the two clients unlocks, copies the value of `next` into g, signals on `sem_server` to unlock the server thread and starts waiting on `sem_print`.

At this point, the server unlocks again, multiplies by 3 the value of g, increments the total number of requests (variable `total_reqs`), signals on `sem_print` and starts waiting again on `sem_server`. The client thread which read the number in the first place therefore unlocks (it is the only one waiting on `sem_print`, the other is waiting on `sem_client`), prints its identifier and the value of g and read the next value from the file to the variable `next`. In case the read operation has been successful (EOF not yet reached), the client unlocks the server, which repeats the previously described operations; if instead the end of the file has been reached, the `while` loop breaks, the client server closes the input file, sets the complete flag, unlocks the server and finally terminates.

When both of the flags have been set, the server breaks the while loop, prints the total number of requests, destroys the semaphores (via calls to `sem_destroy()`) and frees their associated memory (via calls to `free()`), then terminates.