Report on exercise #1

Matteo Corain S256654

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

The proposed solution makes use of two global buffers, called `normal` and `urgent`, with size `BUF_LEN` (set to 20), and four global semaphores to manage the access to those two buffers using a single producer and single consumer protocol (`emptyn`, `emptyu`, `fulln`, `fullu`).

The main thread performs the following operations:

- It initializes the random seed (via the `srand()` function);
- It allocates (via `malloc()`) and initializes (via `sem_init()`) the four semaphores (to `BUF_LEN` for the empty semaphores and to `0` for the full semaphores);
- It creates (via `pthread_create()`) and joins (via `pthread_join()`) the two threads for the producer and the consumer;
- It finally destroys (via `sem_destroy()`) and releases (via `free()`) the semaphores.

The producer thread, which runs the `producer()` function without receiving any data from the caller (its argument is set to NULL) and without returning any data (return value is NULL), loops through the following actions:

- It selects a random number between 1 and 10, multiplies it by 1000000 and stores it in the `tv_nsec` field of the `struct timespec` variable named `sleep_timespec`, which is then passed to the `nanosleep()` system call in order to make the thread sleep for a number of milliseconds between 1 and 10;
- It gets the current timestamp by using the provided `current_timestamp()` function, storing it into the `tstamp` variable, then selects a random number between 0 and 99 to be used for the selection of the buffer to use, storing it into the `bufsel` variable;
- It selects the buffer to use based on the value of the `bufsel` variable; if `bufsel` is lower than `URG_THRES` (set to 80, so that the `normal` buffer is chosen 80% of the times), then it uses the `normal` buffer, otherwise it uses the `urgent` buffer;
- It performs a wait on the empty semaphore related to the chosen buffer (`emptyn` or `emptyu`);
- It puts the value of `tstamp` in the first available position of the selected buffer (by means of the `posn` and `posu` variables, initialized to 0);
- It performs a signal on the full semaphore related to the chosen buffer (`fulln` or `fullu`);
- It updates the value of `posn` or `posu` (by incrementing them and performing a modulo operation by `BUF_LEN`, so that they assume values from `0` to `BUF_LEN-1` in a circular fashion) and `cntn` or `cntu`, which count respectively the number of times each buffer was chosen.

The producer thread loops until the sum of `cntn` and `cntu` is lower than `DATA_LEN`, set to 10000 times; after that, it shows a termination message with some simple statistics on the number of urgent and normal elements produced.

The consumer thread instead, which runs the `consumer()` function again without receiving or returning any data from and to the caller, loops through the following actions:

- It sleeps for 10 milliseconds by using the `nanosleep()` system call, to which the `struct timespec` variable named `sleep_timespec` is passed, whose `tv_nsec` field is set to 10000000;

- It checks if there is data available on the `urgent` buffer by performing a `sem_trywait()` on semaphore `fullu`;
- If the wait operation can be performed without blocking (the function returns `0`), it gets the `tstamp` value from the `urgent` buffer (at index `posu`), posts on semaphore `emptyu` and then updates `posu` and `cntu` similarly to what the producer does;
- Otherwise, it checks if the wait operation can be performed without blocking on the `fulln` semaphore (data is available on the `normal` buffer), and, in that case, it performs the same operations by using that buffer and its associated variables `posn` and `cntn`.

The consumer also loops until the sum of `cntn` and `cntu` is lower than `DATA_LEN`, before showing a termination message with some simple statistics on the number of urgent and normal elements consumed.