Report on exercise #2

Matteo Corain S256654

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

The proposed solution makes use of a global semaphore `s` and a global integer `ret_val` to store the return value for the function `wait_with_timeout()`, mapped on constants `EXIT_NORM` (`0`) and `EXIT_TOUT` (`1`).

The main function checks the number of parameters passed on the command line, transforms the first one to an integer (using the library function `atoi()` and storing it in the variable `tmax`), initializes the random seed for `rand()` to the current time (via `srand()`), allocates and initializes to `0` the global semaphore (checking the correctness of these operations). After that, it creates two threads to run functions `thread_runner_1()` and `thread_runner_2()` (by calling `pthread_create()` twice), joins them (via `pthread_join()`) and finally destroys (via `sem_destroy()`) and frees (via `free()`) the semaphore.

The first thread performs the following actions:

- Selects a random number of milliseconds between 1 and 5 and stores it in `sleep_time`;
- Sets the fields of the `sleep_timespec` structure to reflect the value of `sleep_time`;
- Sleeps the given amount of milliseconds via the `nanosleep()` system call, to which the `sleep_timespec` structure is passed;
- Waits on semaphore `s` via the `wait_with_timeout()` function (whose argument `tmax` is passed to the thread function by the main function), printing the respective termination message based on its return value.

The second thread performs the following actions:

- Selects a random number of milliseconds between 1000 and 10000 and stores it in `sleep_time`;
- Sets the fields of the `sleep_timespec` structure to reflect the value of `sleep_time`;
- Sleeps the given amount of milliseconds via the `nanosleep()` system call, to which the `sleep_timespec` structure is passed;
- Performs a signal on semaphore `s`.

The function `wait_with_timeout()` permits to wait on the passed semaphore `s` for a maximum amount of time `tmax` and returns either `EXIT_NORM` (semaphore unlocked before timeout) or `EXIT_TOUT` (semaphore unlocked for timeout). To achieve this behavior, the function makes use of a POSIX timer to set an alarm with finer granularity with respect to the standard `alarm()` system call, which takes as a parameter a value in seconds. In particular, the function:

- Registers function `sig_handler()` as the signal handler for `SIGALRM`, which sets the return value to `EXIT_TOUT` and performs a signal on the global semaphore;
- Creates a timer via the `timer_create()` system call, passing as an argument a variable of type `struct sigevent`, which describes what to do when the timeout expires; in this case, the timer is configured to send a `SIGALRM` signal at expiration time;
- Starts the timer via the `timer_settime()` system call, which receives as a parameter a `struct itimerspec`, whose two fields are two `struct timespec` indicating the first expiration time (set to `tmax`, with opportune conversions) and the repetition interval (set to `0`);
- Sets the return value to `EXIT_NORM`, then waits on the global semaphore;
- Restores the handler for `SIGALRM` to the default one, destroys the timer and returns `ret_val`.