Report on exercise #2

Matteo Corain S256654

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

The proposed solution makes use of the function `wait_with_timeout()` to wait on the given semaphore specifying a maximum timeout. This function is implemented in the source file `wait_with_timeout.c` (in order to hide some implementation details, such as the global variables `sem_global` and `ret_val`) and exported in the corresponding header file; its return value is mapped on constants `EXIT_NORM` (`0`) and `EXIT_TOUT` (`1`). 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 maximum waiting time (in milliseconds) and the semaphore on which the timed wait should be performed. Additionally, the predefined data structure `struct timespec` has been used throughout the exercise to define the parameters for different functions and system calls; its purpose is to represent timespans with nanoseconds precision, and it contains two fields with the following meanings:

- `tv_sec` (type `time_t`): it represents a number of seconds;
- `tv_nsec` (type `long`): it represents a number of nanoseconds, which should not exceed 1 second.

The main function performs the following actions:

- It checks the number of parameters passed on the command line;
- It transforms the first one to an integer (using the library function `atoi()` and storing it in the local variable `tmax`);
- It initializes the random seed for `rand()` to the current time (via `srand()`);
- It allocates and initializes to `0` the semaphore `s` (via `malloc()` and `sem_init()`, checking the correctness of these operations);
- It prepares the `thread_data` structure of type `thread_data_s`, setting its field to contain a pointer to the semaphore `s` and the value of `tmax`;
- It creates two threads to run functions `thread_runner_1()` and `thread_runner_2()` (by calling `pthread_create()` twice), passing to both of them the `thread_data` structure;
- It waits until both of the threads terminate (via `pthread_join()`);
- It destroys (via `sem_destroy()`) and frees (via `free()`) the semaphore.

The first thread performs the following actions:

- It retrieves the pointer to the semaphore and the value of `tmax` from the `thread_data` structure;
- It selects a random number of milliseconds between 1 and 5 and stores it in `sleep_time`;
- It sets the fields of the `sleep_timespec` structure to reflect the value of `sleep_time`;
- It sleeps the given amount of milliseconds via the `nanosleep()` system call, to which the `sleep_timespec` structure is passed;
- It 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:

- It retrieves the pointer to the semaphore from the `thread_data` structure;
- It selects a random number of milliseconds between 1000 and 10000 and stores it in `sleep_time`;

- It sets the fields of the `sleep_timespec` structure to reflect the value of `sleep_time`;
- It sleeps the given amount of milliseconds via the `nanosleep()` system call, to which the `sleep_timespec` structure is passed;
- It performs a signal on semaphore `s`.

For the implementation of the timeout mechanism for the wait on the semaphore in the function `wait_with_timeout()`, it has been chosen to use a POSIX timer instead of the system call `alarm()`; the usage of a timer, in fact, permits to set the timeout before the alarm with nanoseconds precision, instead of the seconds precision given by `alarm()`. To support this feature, it has been necessary to increase the POSIX compliance level of the code and to link the executable against the real time library (by compiling with GCC flag `-lrt`). The function `wait_with_timeout()` performs the following actions:

- It copies the pointer to the semaphore `s` to the global variable `sem_global`, so that it can be accessed also in the signal handler (signal handlers take as the only parameter the number of the signal which has been received);
- It registers function `sig_handler()` as the signal handler for `SIGALRM`, which sets the global variable `ret_val` to `EXIT_TOUT` and performs a signal on the global semaphore;
- It creates a timer via the `timer_create()` system call, passing as an argument a variable of type `struct sigevent`; this data structure is used to describe what the timer should do when the timer expires: in this case, it has been setup to generate a signal (field `sigev_notify` set to `SIGEV_SIGNAL`) of type `SIGALRM` (field `sigev_signo` set to `SIGALRM`);
- It starts the timer via the `timer_settime()` system call, which receives as a parameter a `struct itimerspec`; this data structure is used to describe the time intervals for the timer expiration: it contains two fields, both with type `struct timespec`, which indicate respectively the time for the first expiration (`it_value`, in this case set to `tmax` with opportune conversions) and for the successive ones (`it_interval`, in this case set to `0` with opportune conversions);
- It sets the global variable `ret_val` to `EXIT_NORM`, then waits on the global semaphore;
- It restores the handler for `SIGALRM` to the default one, destroys the timer and returns `ret_val`.