Text of exercise #2

Matteo Corain S256654

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

Implement a concurrent program in C language, using `Pthreads`, which generates two threads, and then wait for their completion. The first thread `th1` must:

- Sleep a random number of milliseconds `t` in range 1 to 5;
- Print "waiting on semaphore after `t` milliseconds";
- Wait on a semaphore `s`, initialized to `0`, no more than `tmax` milliseconds (`tmax` is passed as an argument of the command line);
- Print "wait returned normally" if a `sem_post(s)` was performed by the second thread `th2` within `tmax` milliseconds from the wait call (or if the `sem_post` call is performed by `th2` before the `sem_wait` call performed by th1);
- Otherwise, it must print "wait on semaphore `s` returned for timeout";
- Terminate.

The second thread `th2` must:

- Sleep a random number of milliseconds `t` in range 1000 to 10000;
- Print "performing signal on semaphore `s` after `t` milliseconds";
- Terminate.

For the first thread, you must implement and use a function with prototype:

```
int wait_with_timeout(sem_t *s, int tmax)
```

Which, using the appropriate system calls for the management of semaphores and `SIGALARM` signals, allows you to define the maximum time that a process can be blocked on the semaphore `s` queue before it is unblocked, and can proceed regardless of a call to `sem_post(s)`. Function `wait_with_timeout` returns a flag set to 1 if a timeout occurred.

For sleeping less than a second use `nanosleep` system call (`man nanosleep`):

```
#include <time.h>
int nanosleep(const struct timespec *req, struct timespec *rem);
```