Lab 2.1 (Semaphores and signals)

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

a. Sleep a random number of milliseconds **t** in range **1** to **5**
b. Print "**waiting on semaphore after t milliseconds**"
c. Wait on a semaphore **s,** initialized to 0, no more than **tmax** milliseconds (**tmax** is passed as argument of the command line),
d. 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**".
e. Terminate

The second thread **th2** must:

a. Sleep a random number of milliseconds **t** in range **1000** to **10000**
b. Print "**performing signal on semaphore s after t milliseconds**"
f. 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.

Lab 2.2 (Synchronization with semaphores)

Implement a concurrent program in C language, using Pthreads, which generates **n** threads **th_A** executing code **A**, and **2*n** threads **th_B** executing code **B**.
The main thread takes argument **n** from the command line, and terminates without waiting the termination of the threads it has created.

Both threads simply print their corresponding character, followed by their order of creation number (from **0** to **n-1** for **th_A**, from **0** to **2*n-1** for **th_B**).
After two **Bs** and an **A** characters have been printed, in whichever order, the last printing thread must print the **'\n'** character.

Example: **BBA 4**

```
A1 B2 B3
B4 B5 A0
B1 A2 B0
A3 B7 B6
```

Lab 2.3 (Synchronization with semaphores)

Implement a concurrent program in C language, using Pthreads, which generates two threads **th_1** and **th_2**.
The main thread takes an argument **n**, and waits the termination of the threads it has created.

Thread **th_1** loops **n** times printing **A1** then **B1** in sequence on the same line.
Thread **th_2** loops **n** times printing **A2** then **B2** in sequence on the same line.
However, **B1** and **B2** must be printed only after **A1** and **A2**, and the last thread must also print the newline character

Example: **AB12 4**

```
A1 A2 B2 B1
A2 A1 B2 B1
A1 A2 B1 B2
A1 A2 B2 B1
```