Lab 3.1

Write a C program using Pthreads that tests the **sem_trywait** system call on a Producer & Consumer problem using two buffers.
In particular, the main thread must create a Producer and a Consumer thread, and must wait for their termination.
The Producer thread

- loops **10000** times sleeping a random number of milliseconds (**1-10**),
- fills a variable **ms** with the current time in milliseconds, using the function

```
long long current_timestamp() {
  struct timeval te;
  gettimeofday(&te, NULL); // get current time
  long long milliseconds = te.tv_sec*1000LL +   te.tv_usec/1000;
  return milliseconds;
}
```

- selects randomly a buffer **urgent** or **normal** in which it will put the value of **ms**. It must select the normal buffer **80%** of the times.
- prints the message: "**putting** <ms> **in buffer** <buffer>", where <ms> is the value, and <buffer> is either **urgent** or **normal**
- puts the value of **ms** in the selected buffer
- signals on a semaphore that something has been produced

The Consumer thread loops

- sleeps **10** milliseconds
- waiting that something has been produced
- tries to get an **ms** from the **urgent** buffer, but if it is empty it can proceed to get it from the **normal** buffer
- prints the value of **ms** and the buffer identity (**urgent** or **normal**) from which it has got this value

Lab 3.2

Write a C program using Pthreads that implements the Producer & Consumer protocol, where the Producer thread produces and puts on a shared buffer **`10000`** integer numbers (from **`0`** to **`9999`**), and the Consumer thread gets and prints the received numbers (see the example posted in the course site).
You must solve the problem implementing the semaphore functions:
`sema_init, sema_wait, sema_post`
using **conditions**.
You cannot use standard semaphores.

Lab 3.3

Write a C program using Pthreads that sorts the content of a **binary** file including a sequence of integer numbers. Implement a threaded quicksort program where the recursive calls to **`quicksort`** are replaced by threads activations, i.e. sorting is done, in parallel, in different regions of the file.
Use **`mmap`** to map the file as a vector in memory.

If the difference between the **`right`** and **`left`** indexes is less than a value **`size`**, given as an argument of the command line, sorting is performed by the standard **`quicksort`** algorithm.

This is a sequential recursive implementation of the quicksort algorithm.

```
void quicksort (int v[], int left, int right) {
  int i, j, x, tmp;
  if (left >= right)   return;
  x = v[left];
  i = left - 1;
  j = right + 1;
  while (i < j) {
    while (v[--j] > x);
    while (v[++i] < x);
    if (i < j)
      swap (i,j);
  }
  quicksort (v, left, j);
  quicksort (v, j + 1, right);
}

void swap(int i, int j){
  int tmp;

  tmp = v[i];
  v[i] = v[j];
  v[j] = tmp;
}
```