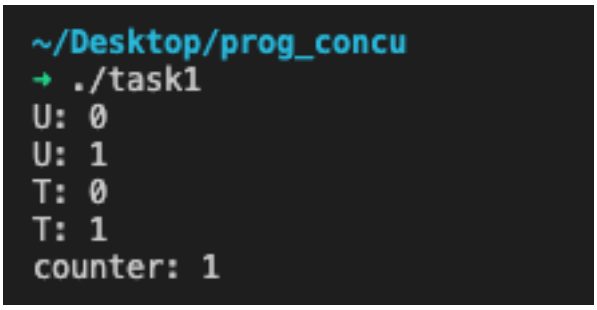
Victor Barbe 403715
Pierre-louis Gaucher 403783

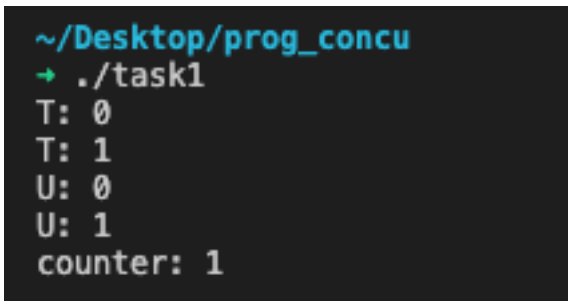
Programación concurrente

1) Multithreads

In the first code of multithreads, there are two functions which are defined working with two different threads. The goal of each function is to increment the value of the counter which is defined outside of the functions.

In this code, the two threads which are working concurrently. They start nearly at the same time, and the two functions works parallelly. There is nothing that makes the execution of one thread complete before another starts. This means that the two threads are trying to increment the same value (the counter) at the same time. We can also see that depending on the execution, one thread can start before the other one.





In the end, since the two threads incremented the same value at the same time, when we print the value of the counter, we get 1 when we were supposed to get 2.

2) Lock

In the second code of locks, there are also two functions which are defined working with two different threads. However, this time we will be using a structure of type pthread_mutex_t that implement a behavior based on the Pthread mutexes. The goal of this structure is to implement a serialized behavior.

This means that once a thread starts, there won't be a possibility for another thread to be functioning at the same time. In our code, this means that we defined two threads, but they will be working one after the other. Once one of the threads starts, it goes all the way to completion and increments the value by one. After the 1st thread is done, the second starts and increments the value by one as well. So, in the end, since the two threads worked one after the other, the value of the counter is 2 as expected.

Just like the first time, we cannot know before the start which thread is going to execute first. However, once one has started, it completely locks the other thread which means no matter the order of execution we always get 2 as a result.