403783 Pierre louis Gaucher

403715 Barbe Victor

Compared to the code of last week, here we will use semaphores. We still create two functions thread\_u and thread\_t, which goal will be to take a counter and increment the value by one. As we saw last week if we were to just create two threads in the main and launch them one after the other, the counter value returned at the end would be 1 instead of two because the threads wouldn’t wait for the other one to finish.

After that in the second code we managed to get 2 as a return value for the counter using lockers that prevent two threads from running at the same time.

This time, we will be using semaphores. In the main, before using the two threads we initialize a semaphore that will be shared between two threads.

In the two functions thread\_u and thread\_t, we add a “sem\_wait()” at the beginning of the function and a “sem\_post” at the end of the function.

If Sem\_wait is positive it decrements the value of the sem, if it isn’t It doesn’t allow for the execution of the following code.

Sem\_post increments the value of the semaphore by one.

The use of those two functions makes sure that the two threads will not be running at the same time. If the value of the sem is equal to one, it will decrement at the beginning of the code to make sure the other thread isn’t running. Once the code is finished, the use of sem\_post will bring it value back to one so the other can thread can run after.

In the end using the semaphores, we will get a counter value of 2 at the end, because the threads ran one after the other properly.

Une image contenant texte

Description générée automatiquement