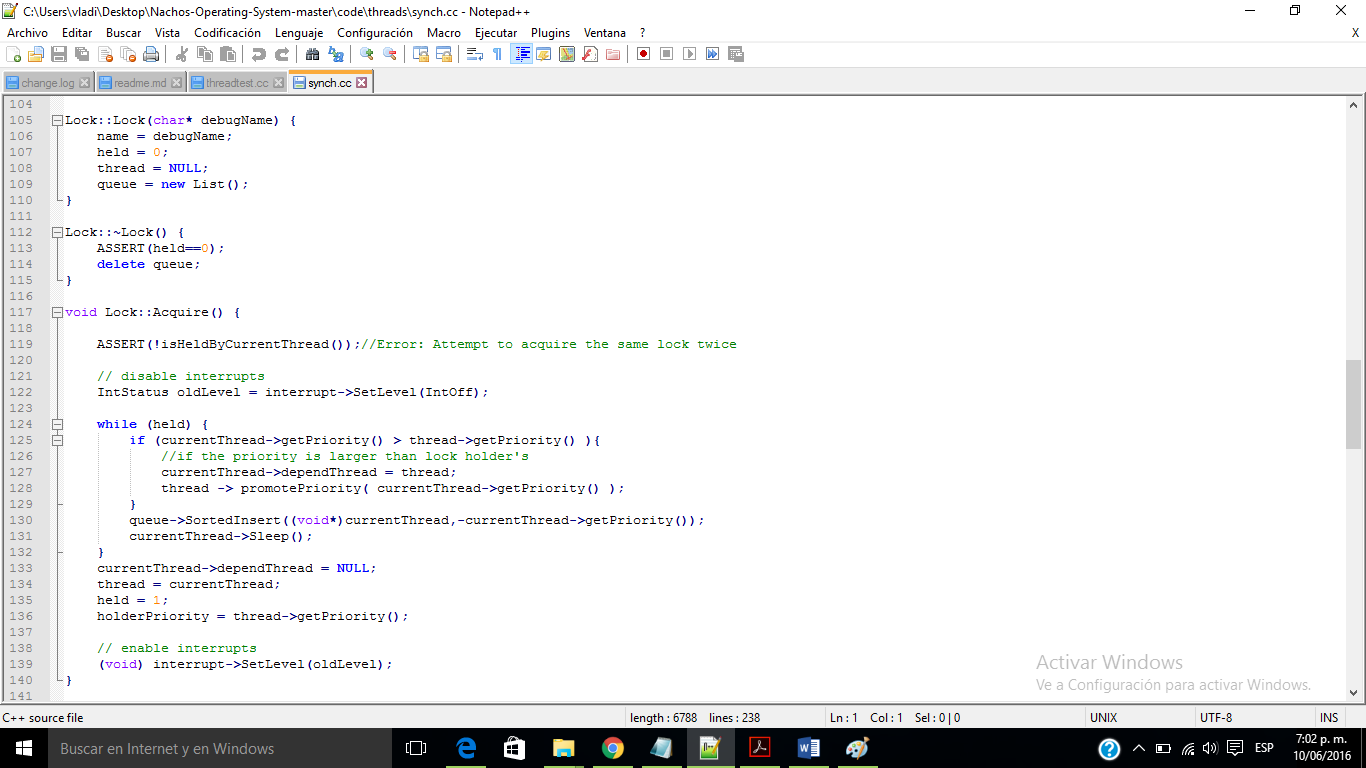
Examen Parcial

* LLUQUE DELGADO , Paul
* MEJIA TASAYCO , Juan Jose
* TORRES CERRON , Vladimir
* URBINA QUISPE, Jaime
* Anccasi Huiza Edda Juliet

Problema 1



We implemented acquire and release using interrupts, asserts, wait and

ready queue. Similarly, condition variables use interrupts, wait and ready

queues, and locks. Instead of appending to the wait queue, we use

SortedInsert() to prioritize the threads.

Test switch case numbers used:

2: Expects to work successfully

3: Assert should fail: Cannot delete a held lock by own thread

4: Assert should fail: Cannot delete a held lock by another thread

5: Assert Should fail: Cannot wait without holding a lock

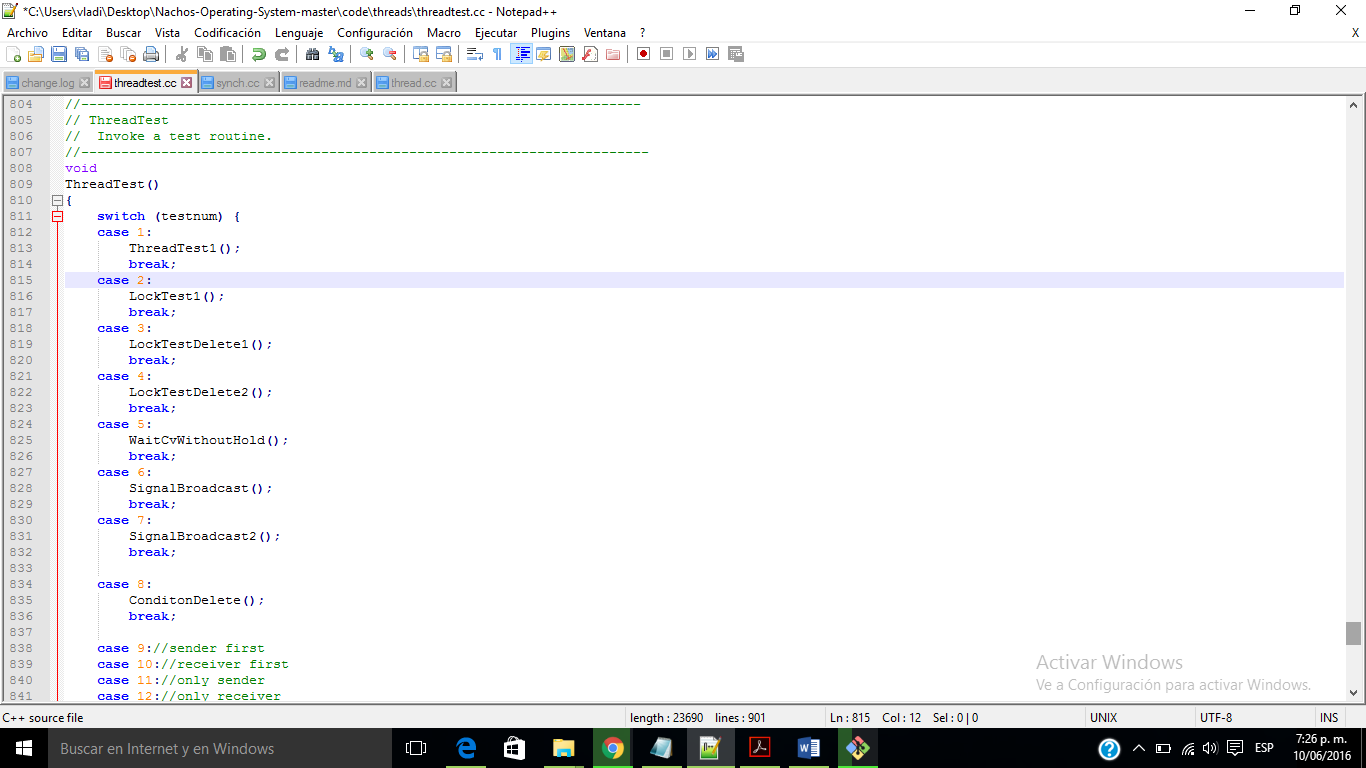
6: Expects to work successfully

7: Expects to work successfully

8: Assert fails: Cannot delete while threads on queue



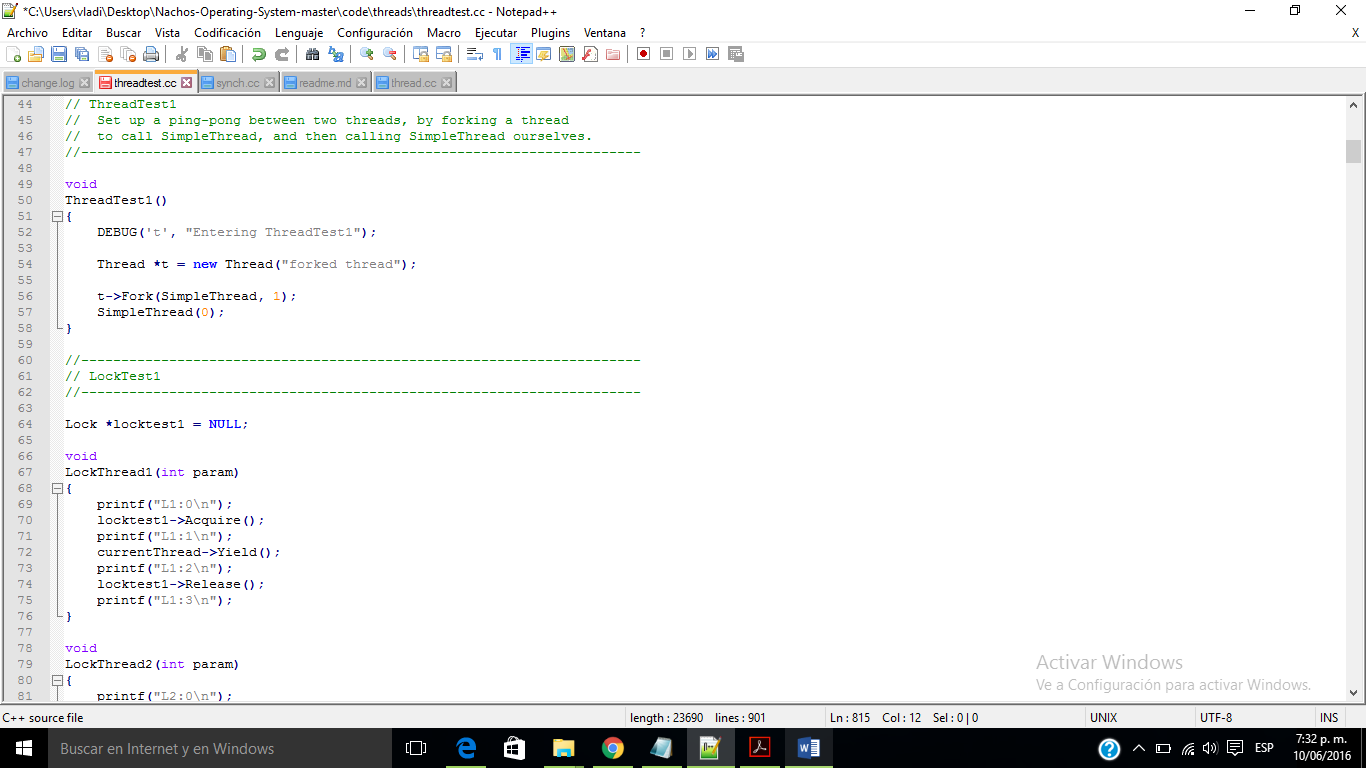
Opciones ingresar casos de la forma: “ ./nachos -q # “



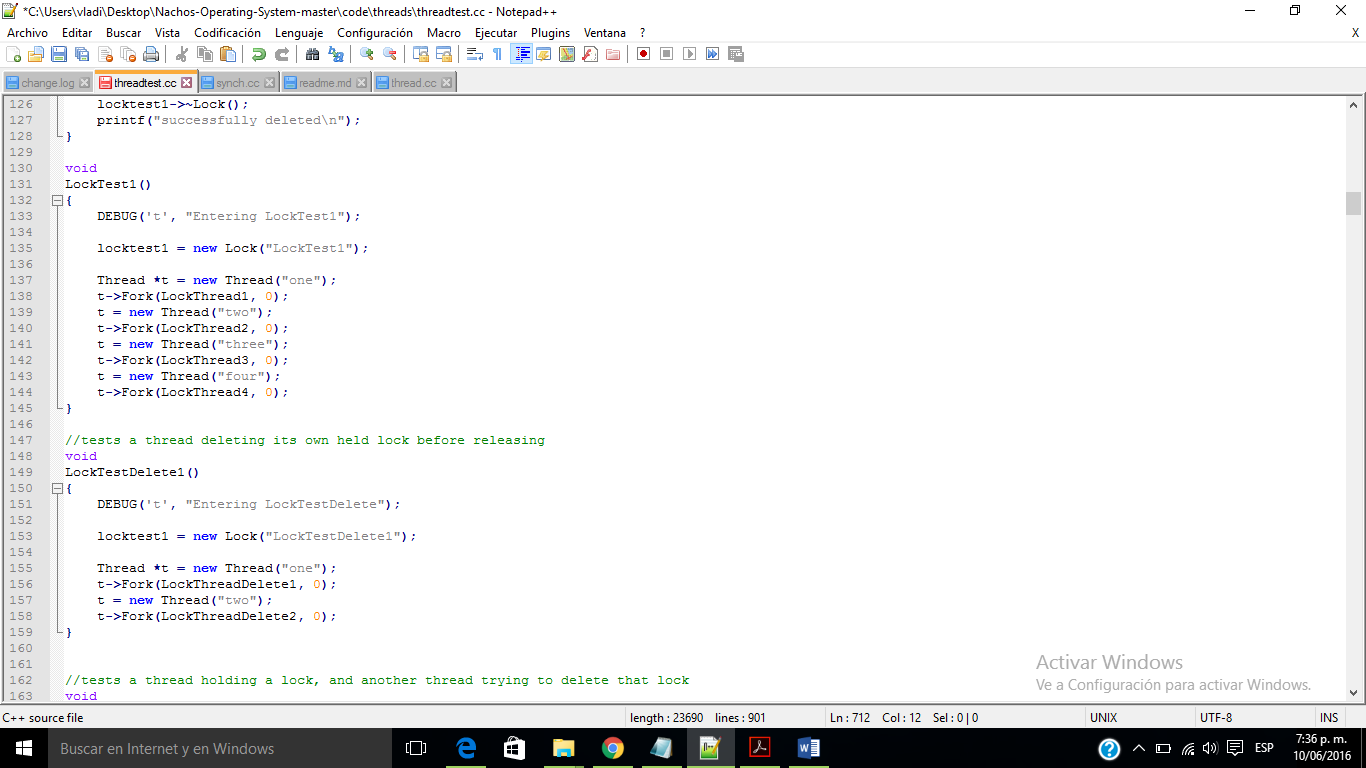
El código de las funciones que llaman.

* Se llama ingresando el código:

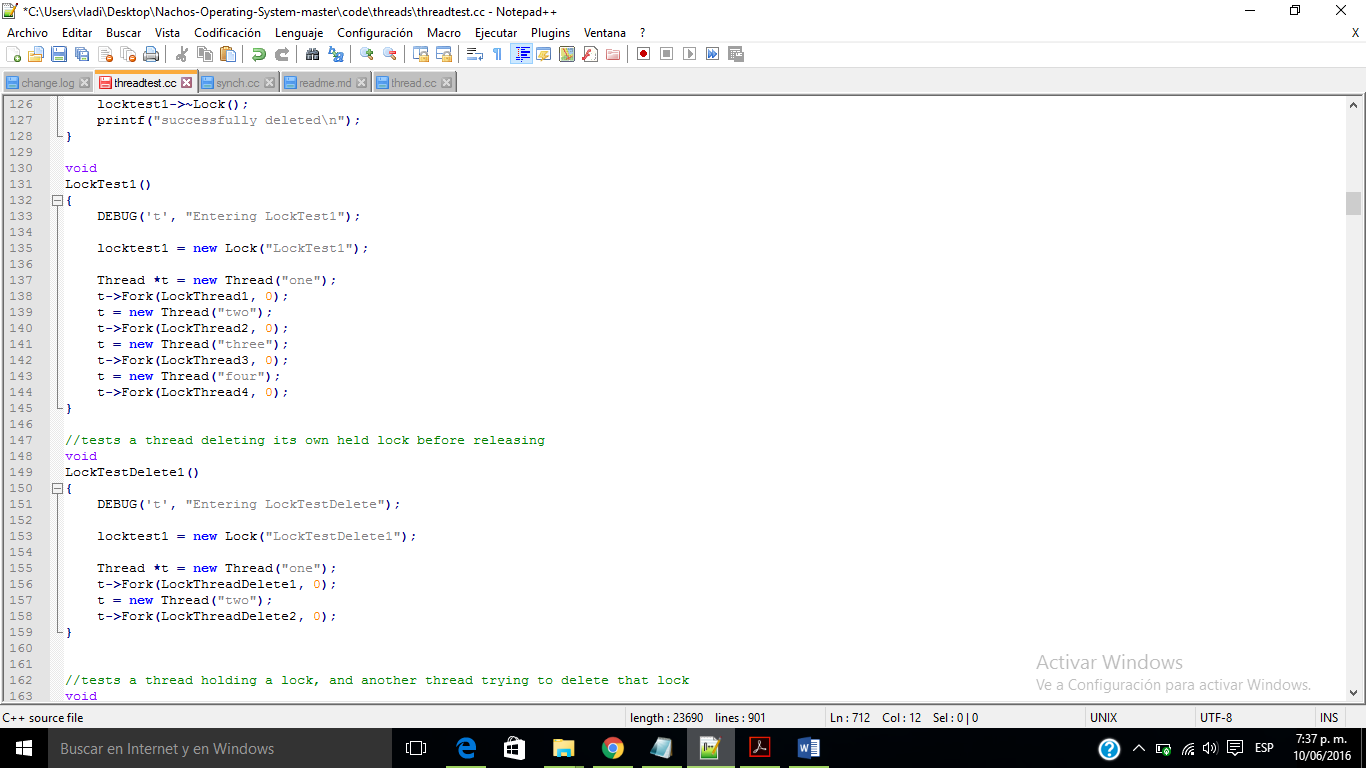
./nachos -q 1



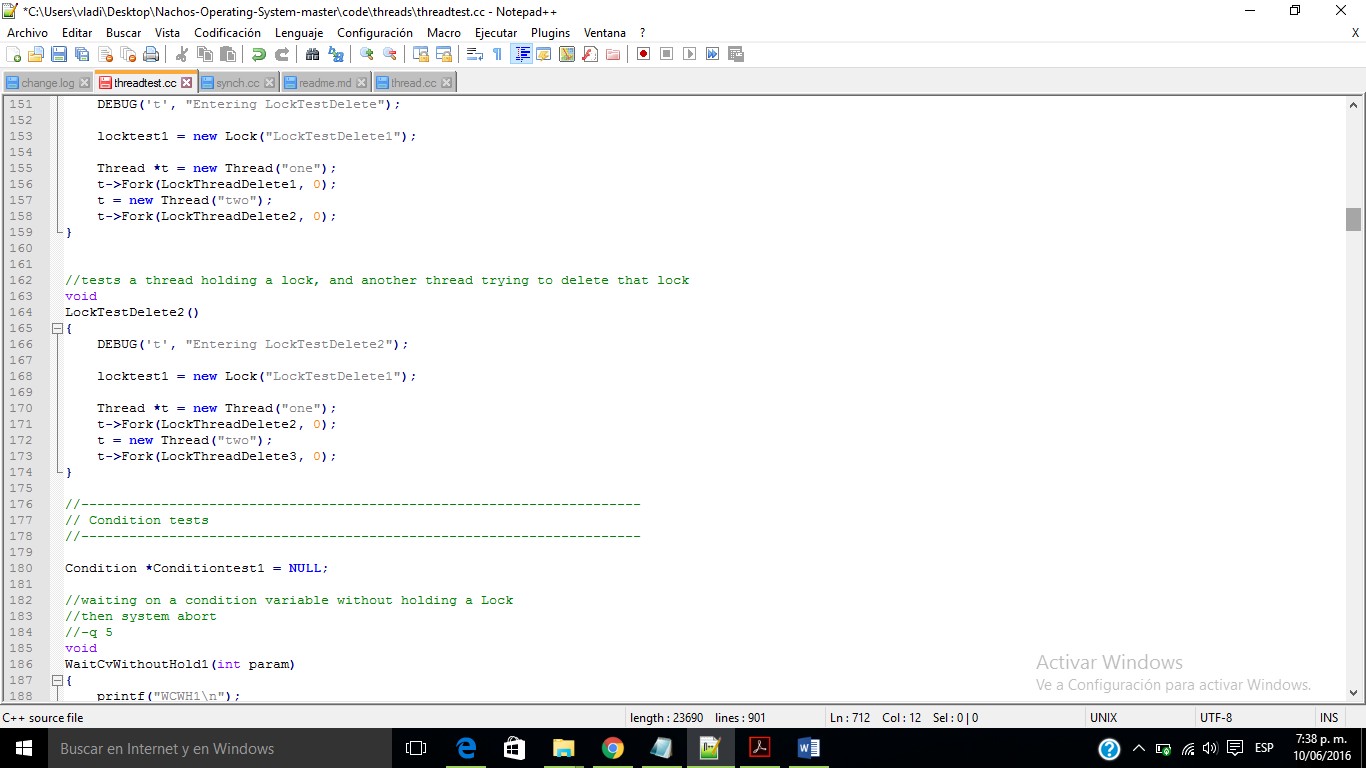
./nachos -q 2



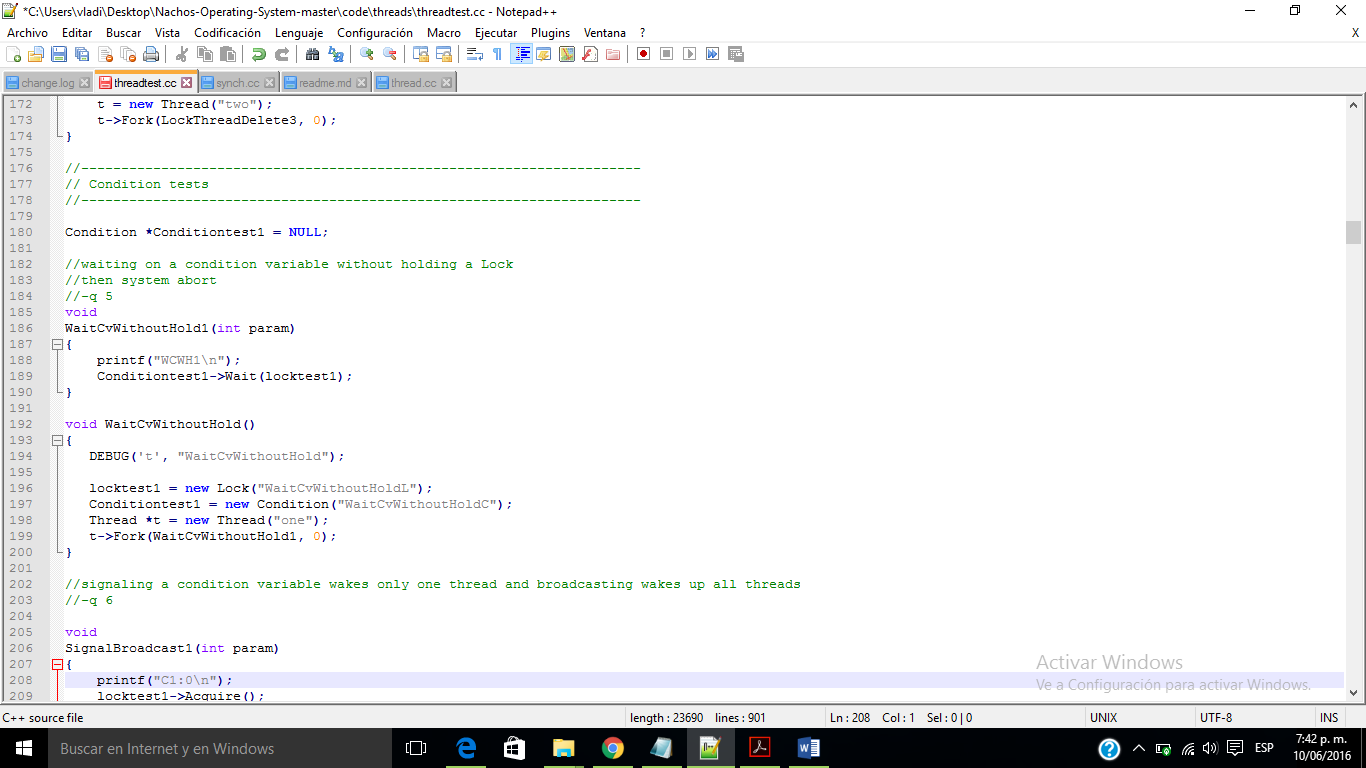
./nachos -q 3



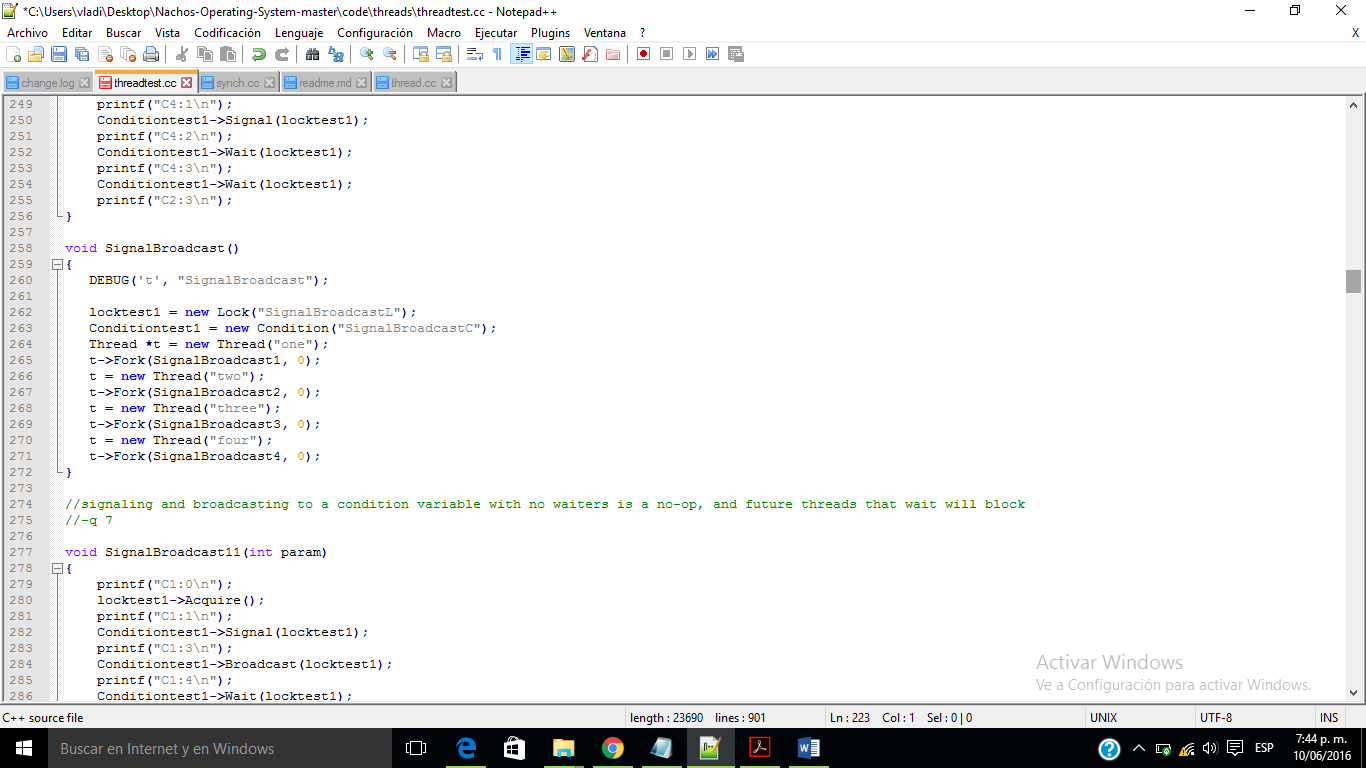
./nachos -q 4



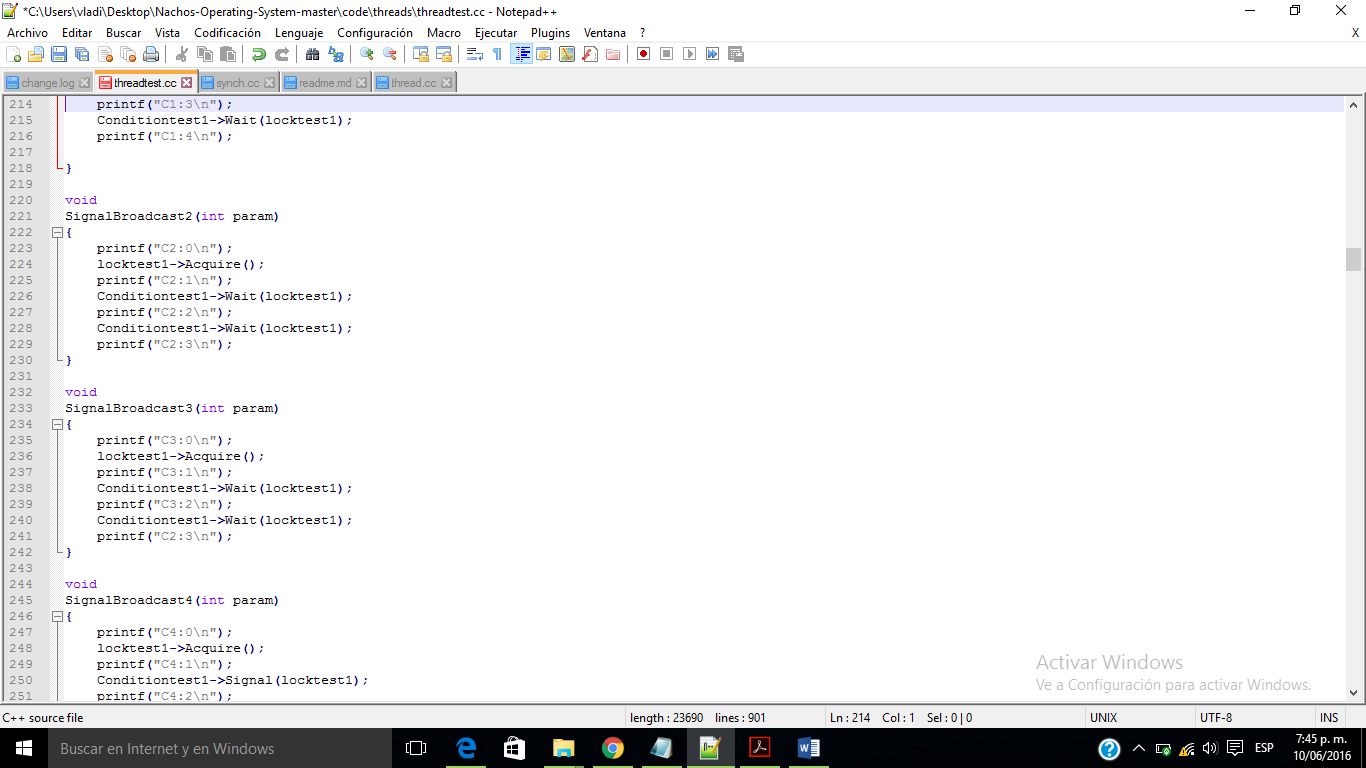
./nachos -q 5



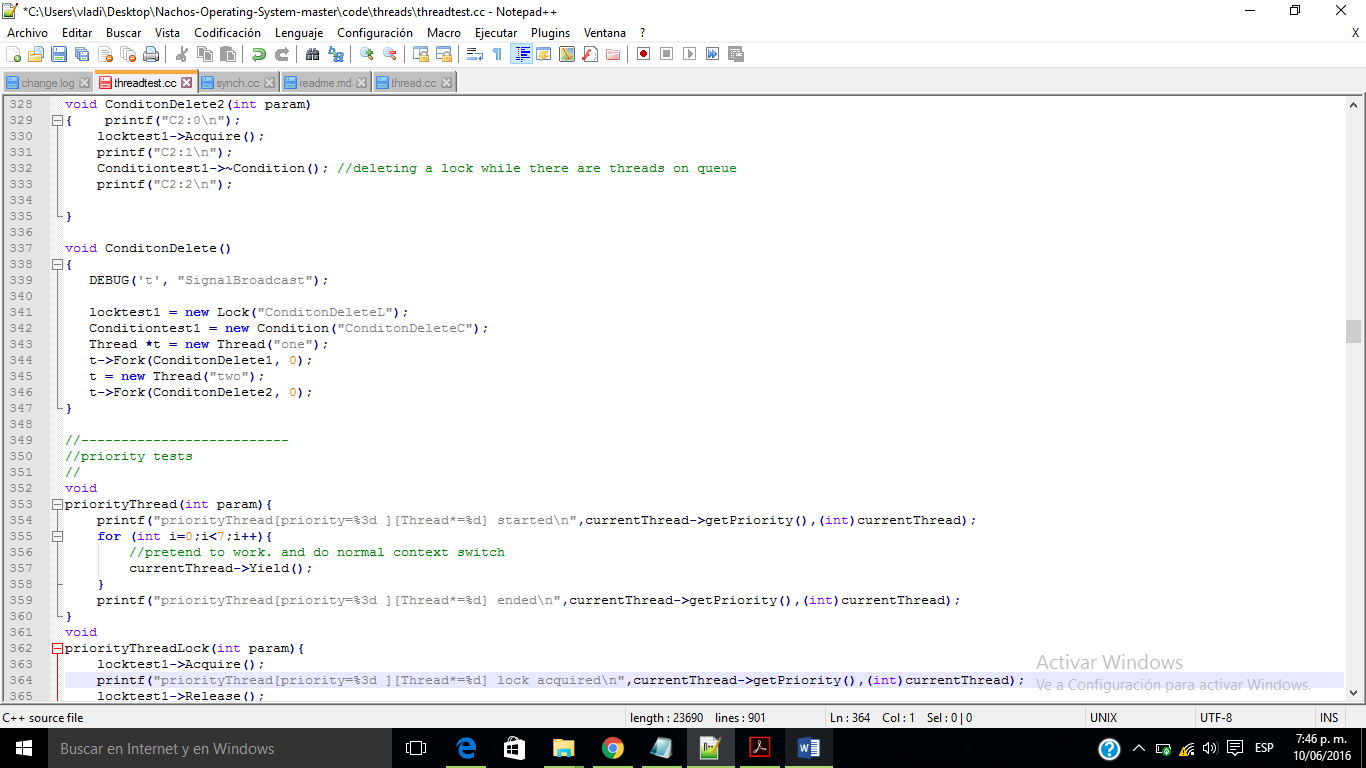
./nachos -q 6



./nachos -q 7

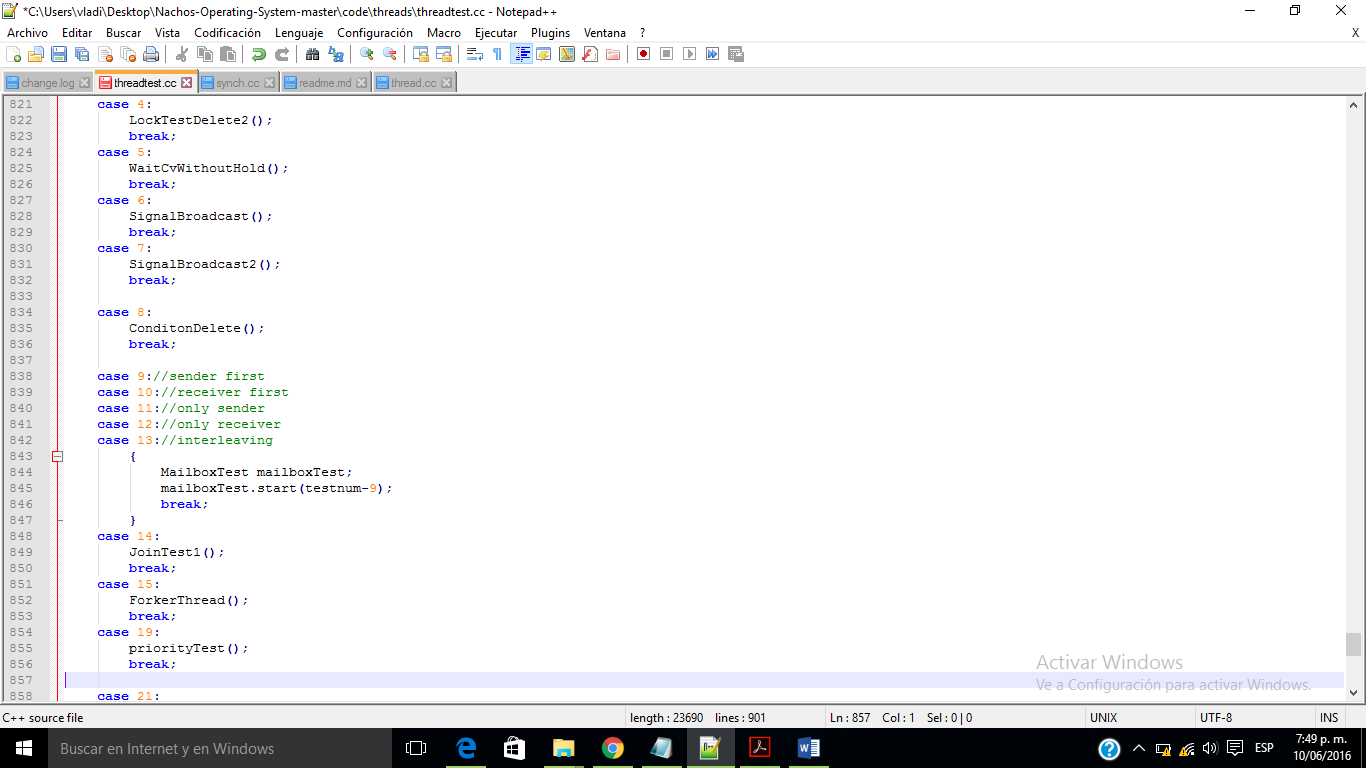


./nachos -q 8



Problema 2:

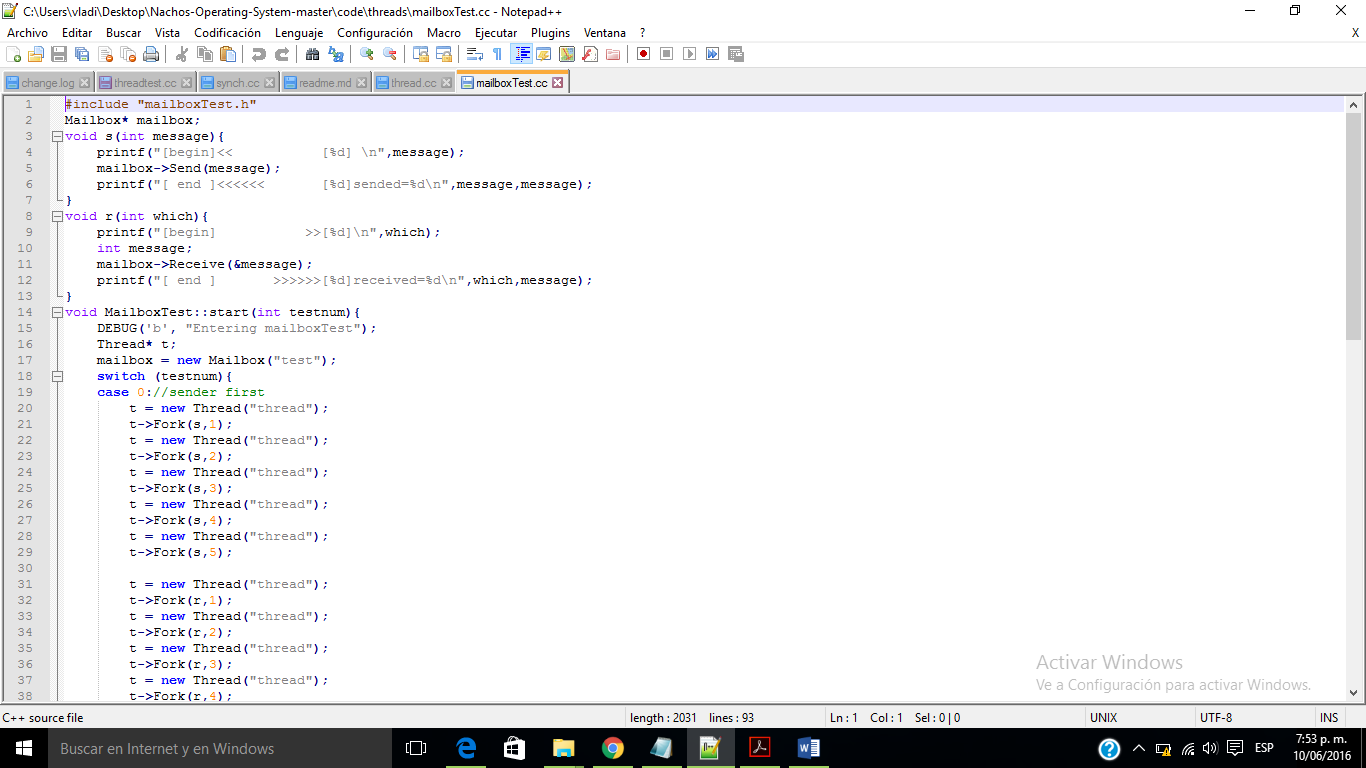
El problema 2 es del caso 9 al 13

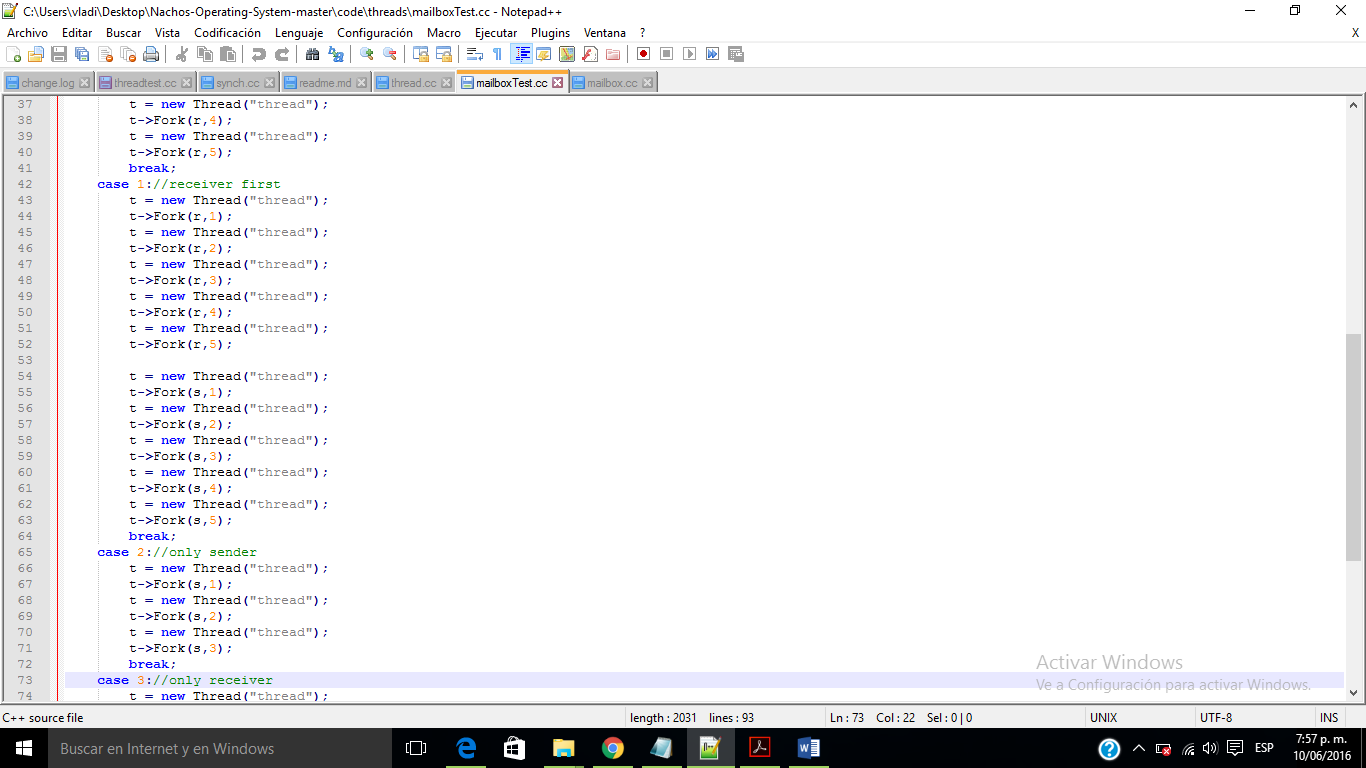


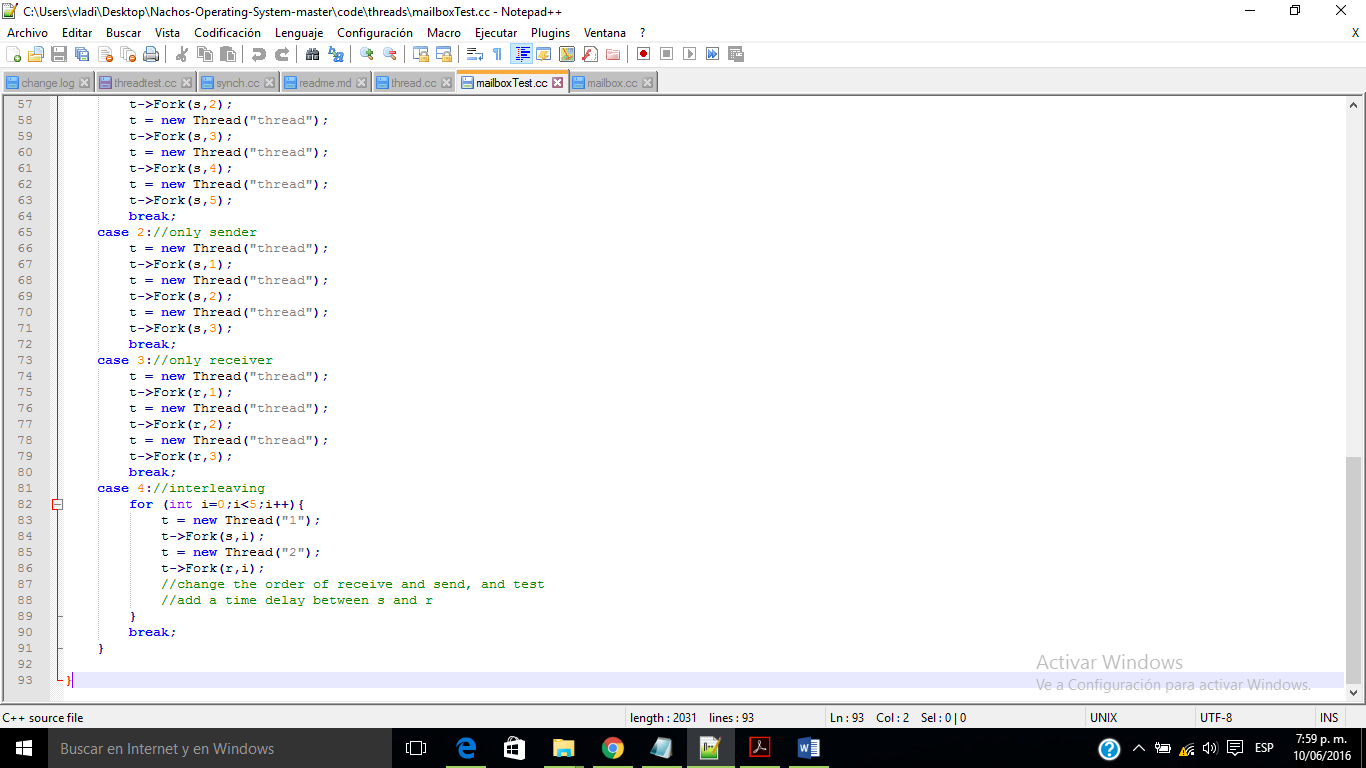
./nachos -q 13

Se esta instanciando el MailboxTest y se inicia.

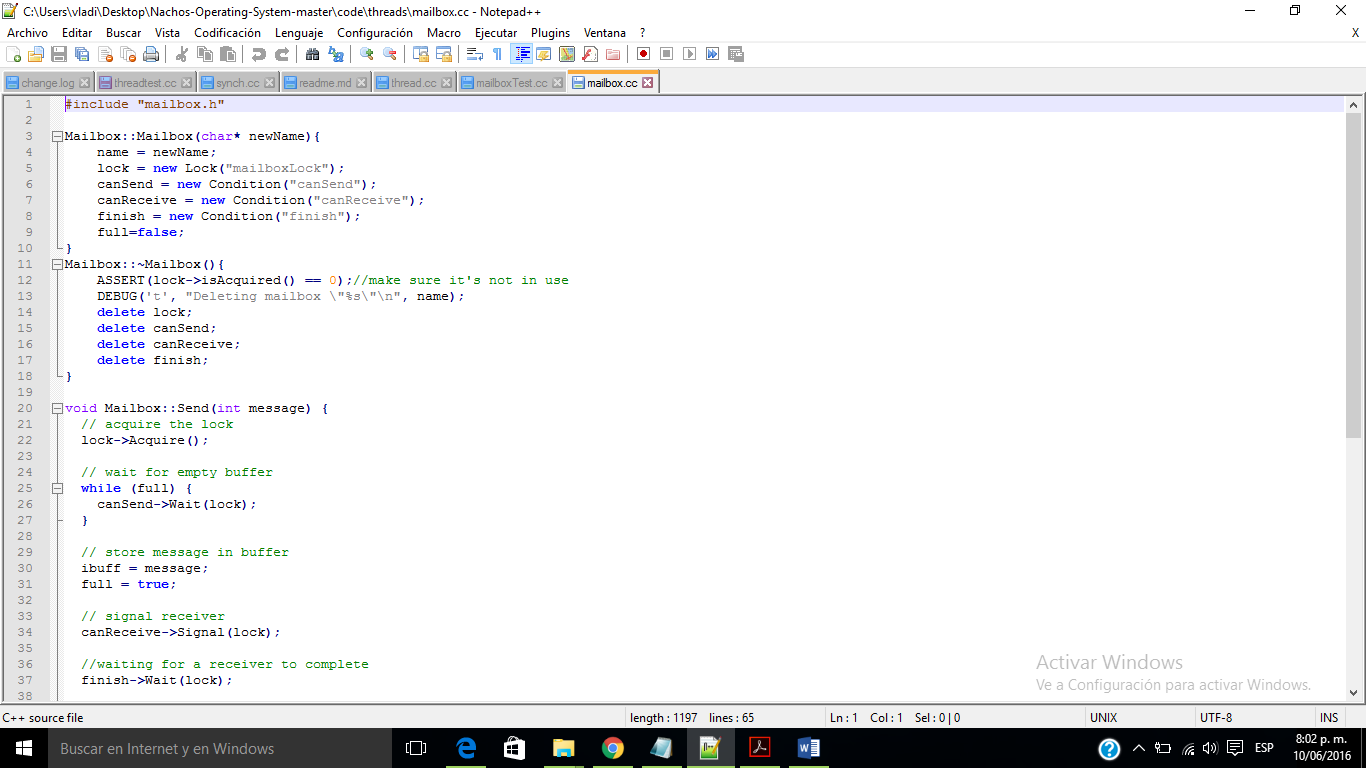
El código para MailboxTest ES:

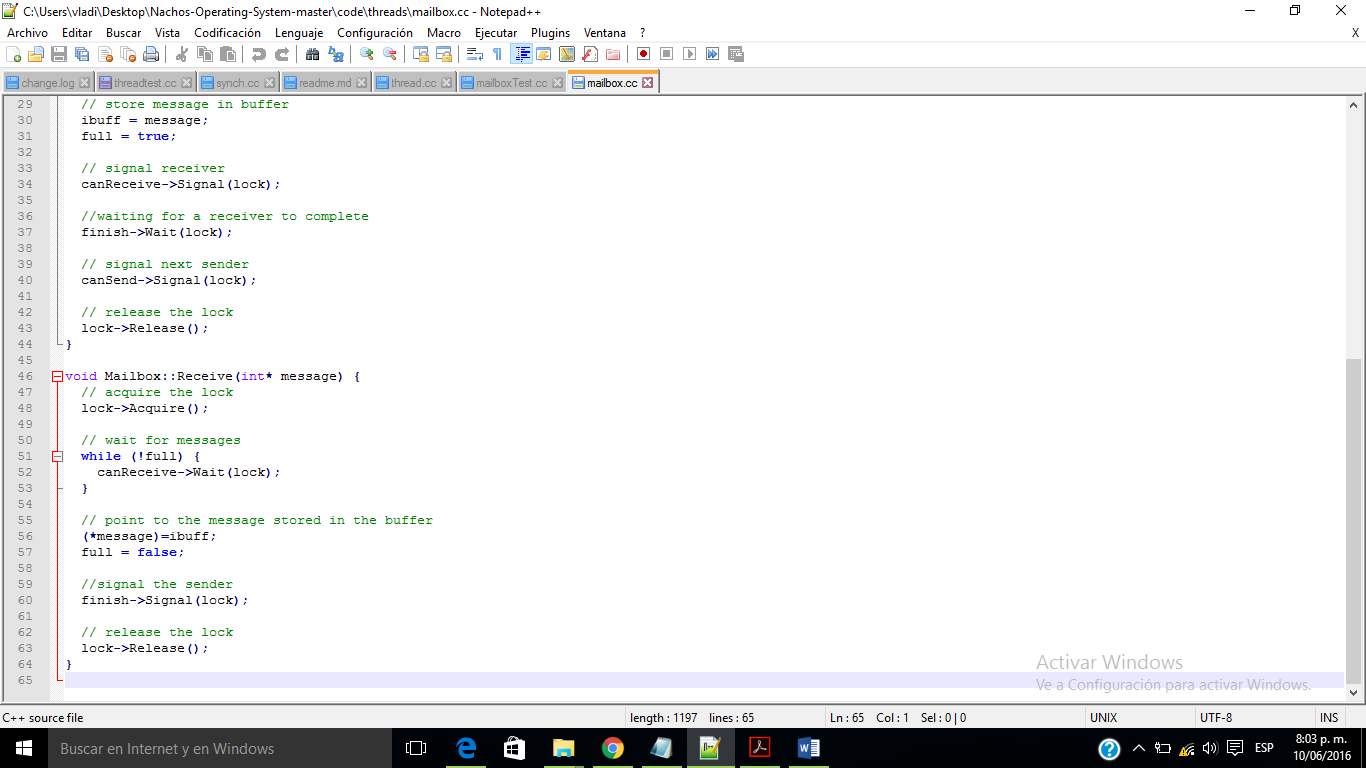






El código para el Mailbox Es:

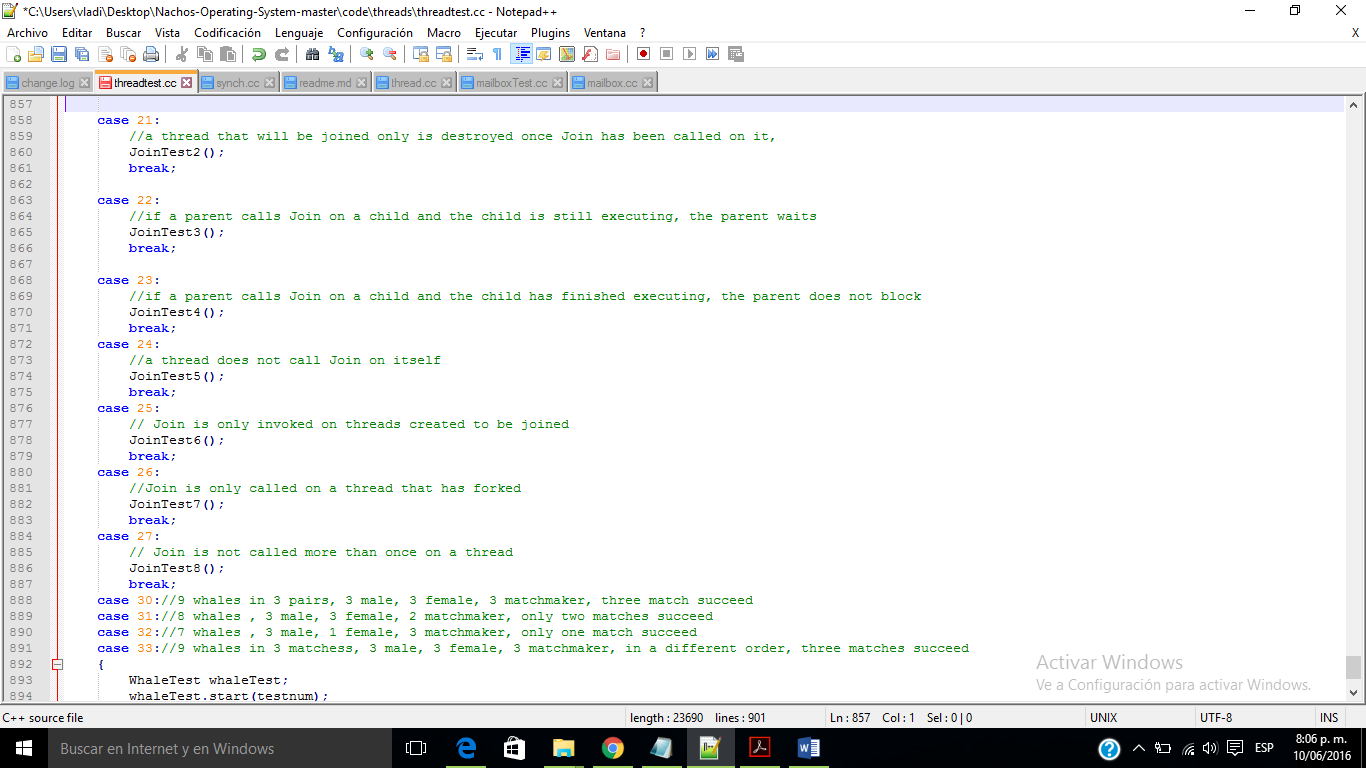




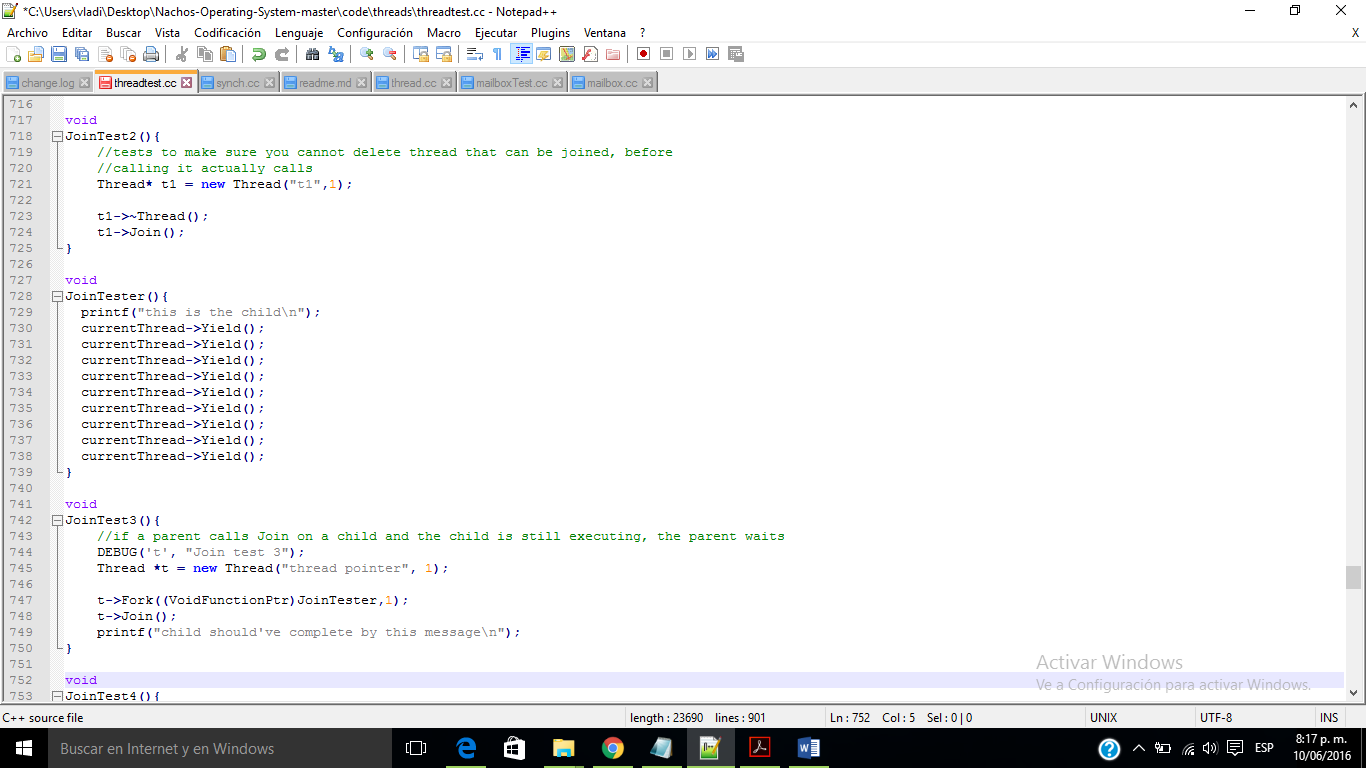
Problema 3:

Se esta instanciando el MailboxTest y se inicia.

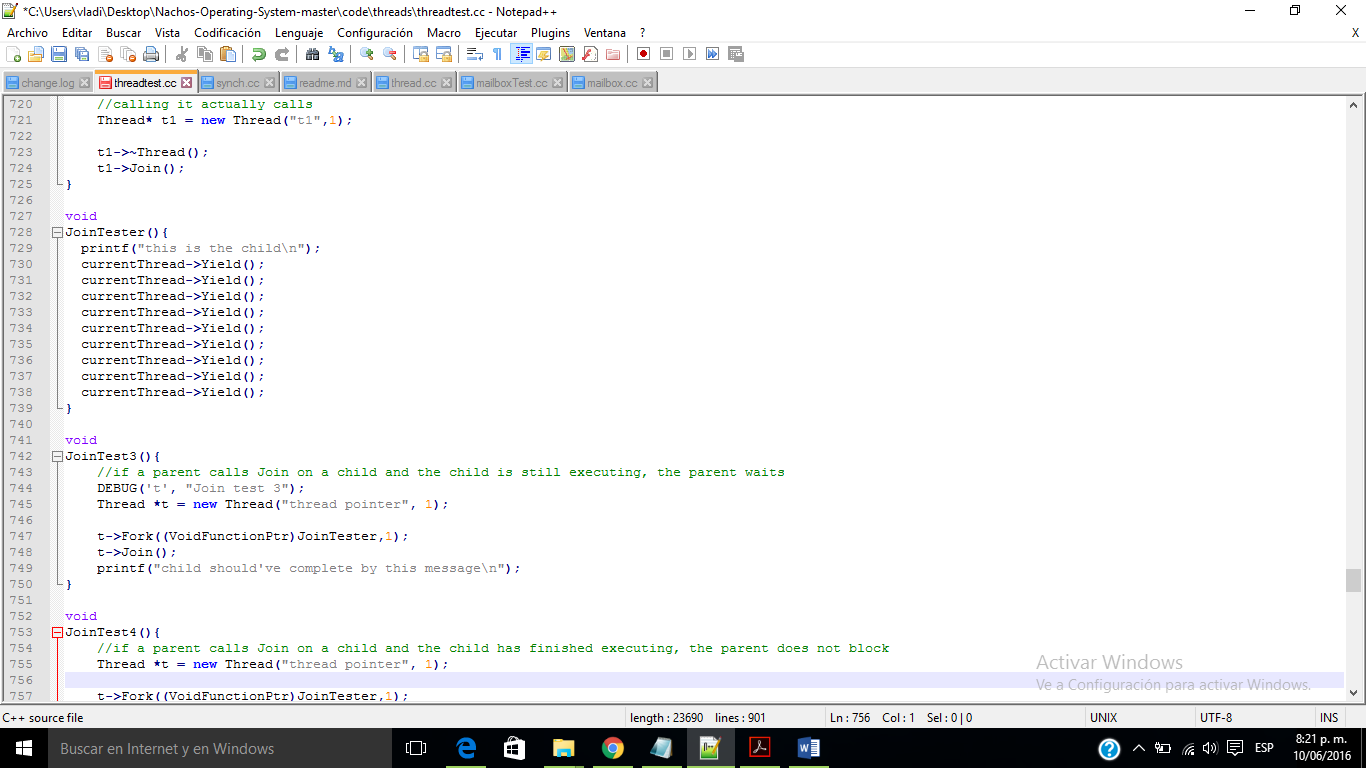
El código para MailboxTest Es:

|

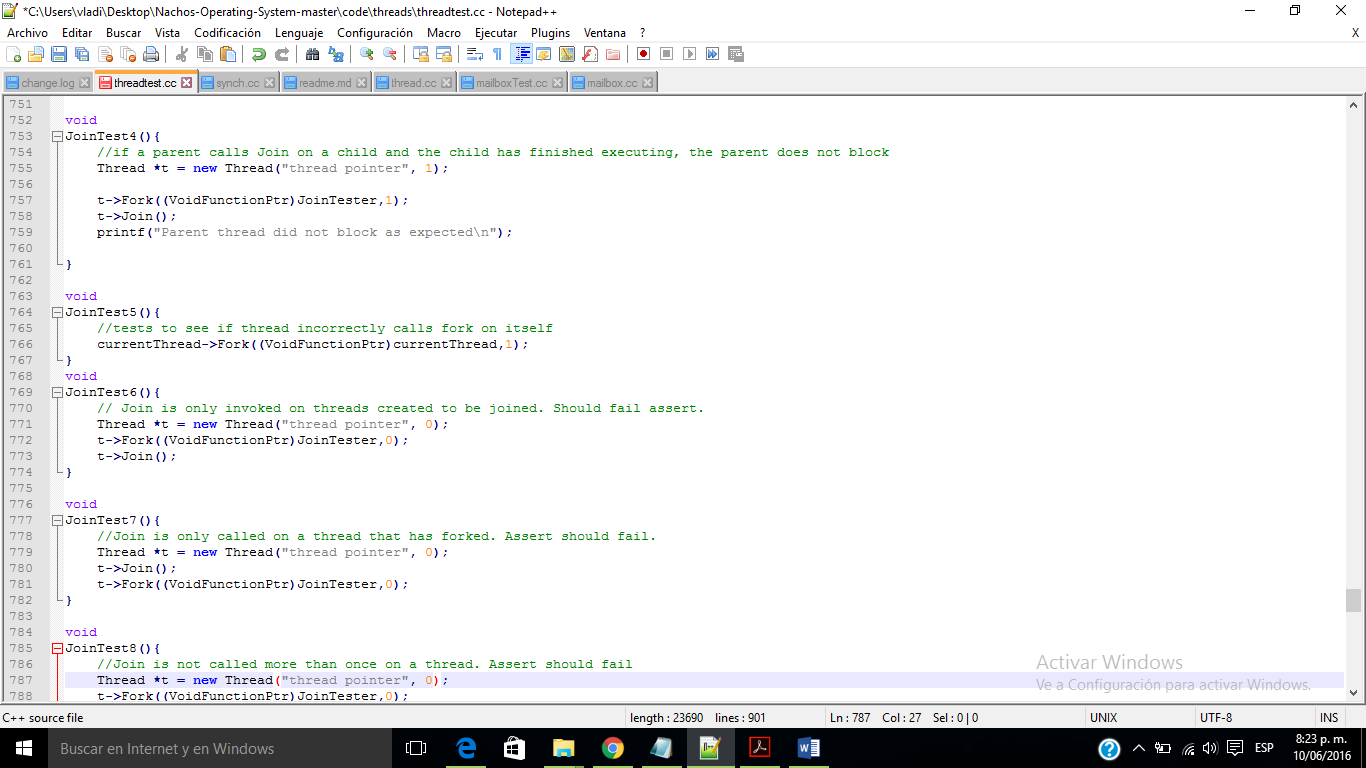
./nachos -q 21



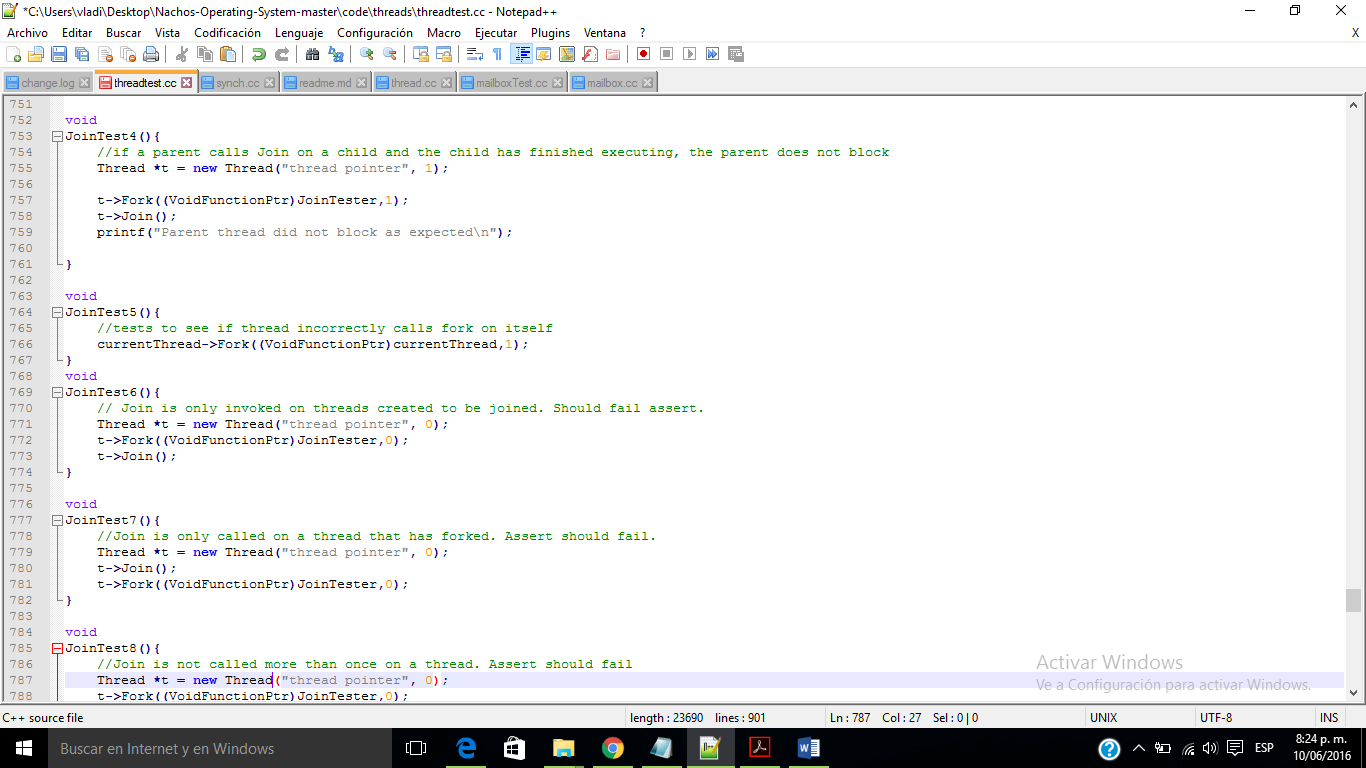
./nachos -q 22



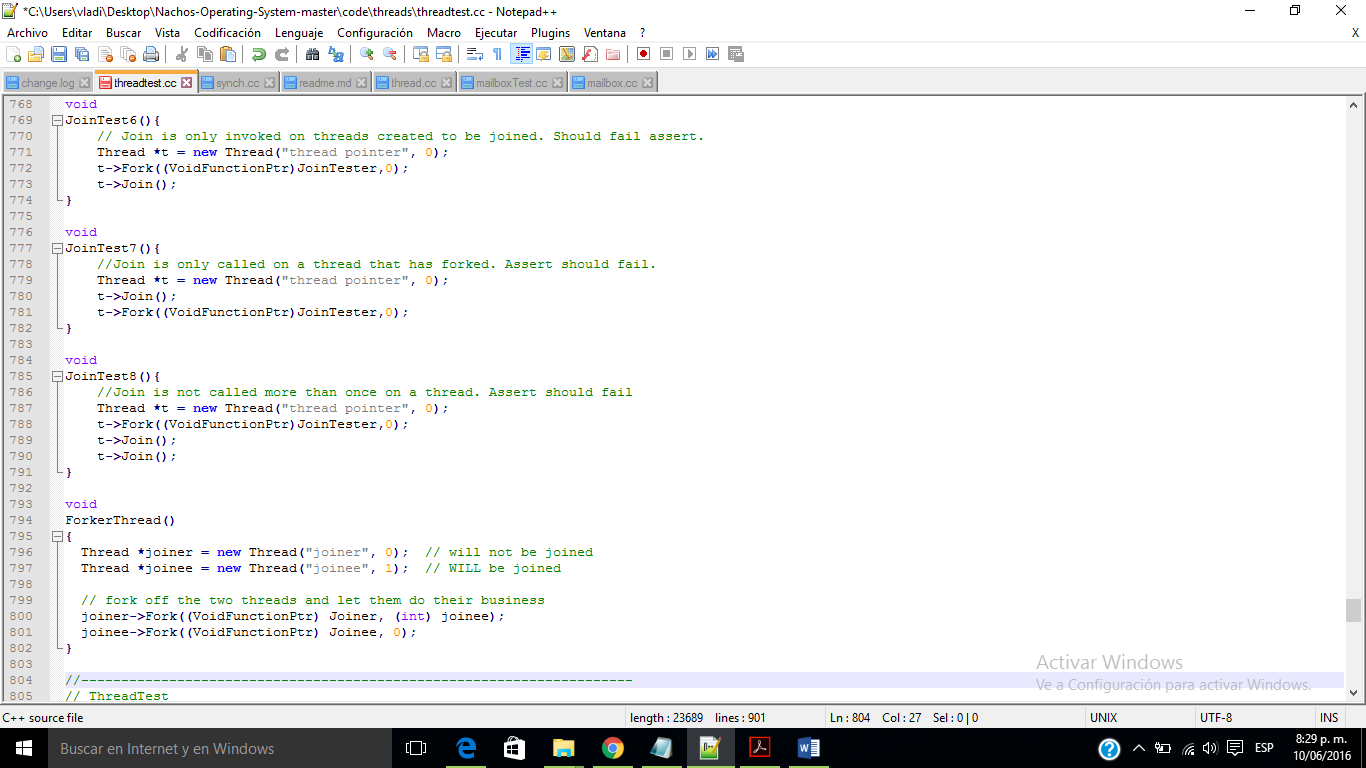
./nachos -q 23



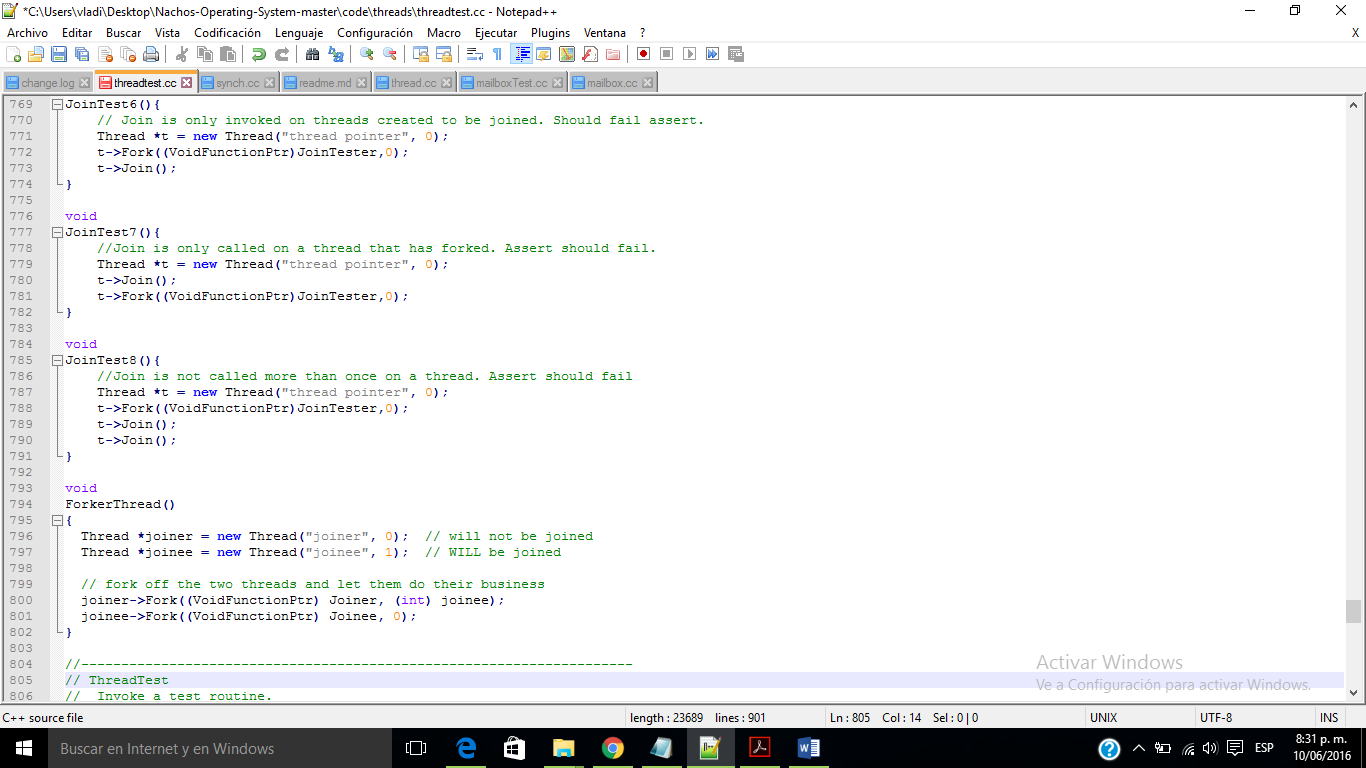
./nachos -q 24



./nachos -q 25



./nachos -q 26



./nachos -q 27



Test switch case numbers used:

19: Expects to work successfully. Outputs for all test cases will be display on

the screen after running "./nachos -q 19".

Those test cases include:

<1> Priority test for scheduler. When Thread::Yield() is called, the scheduler

always run the thread with the highest priority, or switch between threads that share

the highest priority.

<2> Priority test for synch primiives Lock.

<3> Priority test for Semaphore.

<4> Priority test for Condition Var, waked up by Signal();

<5> Priority test for Condition Var, waked up by Broadcast();

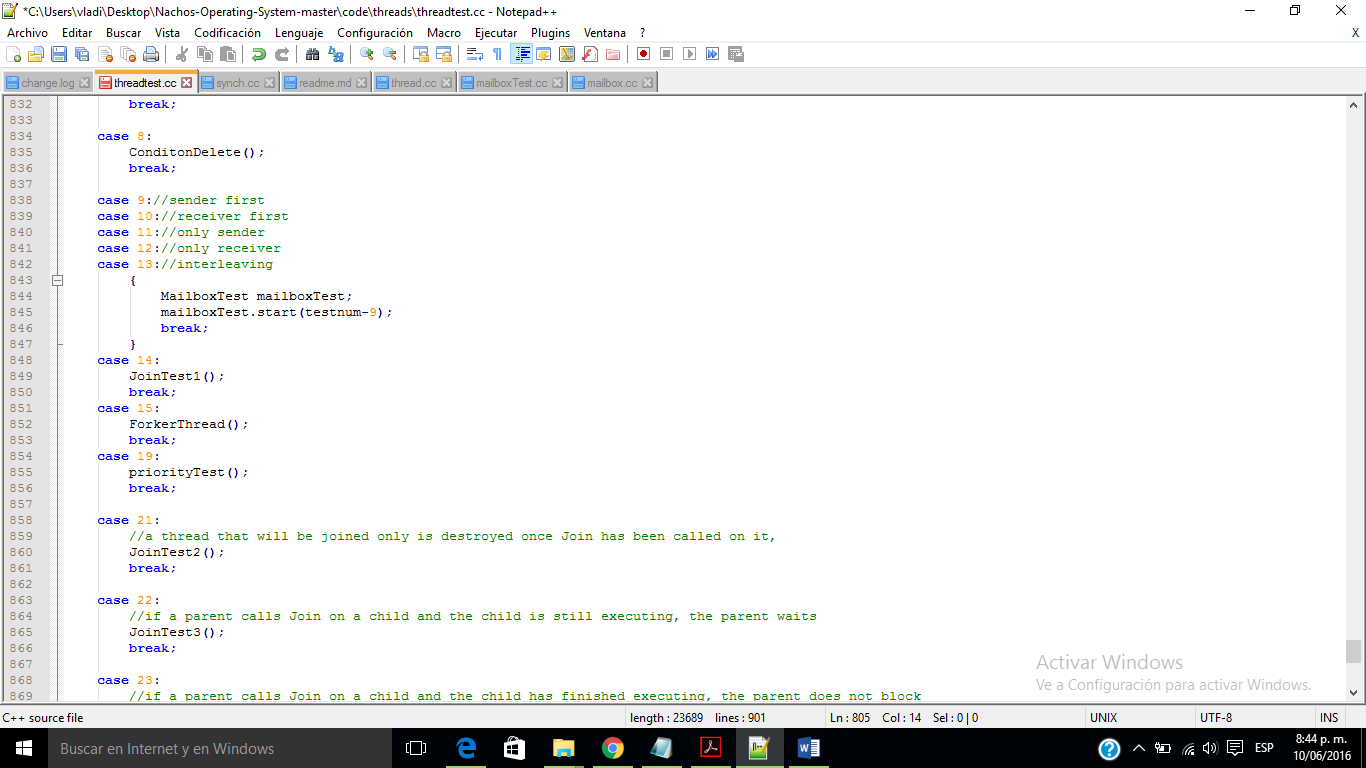
<6> [Extra Credit] There are 5 threads in this case. The lock-holding thread is successfully promoted,

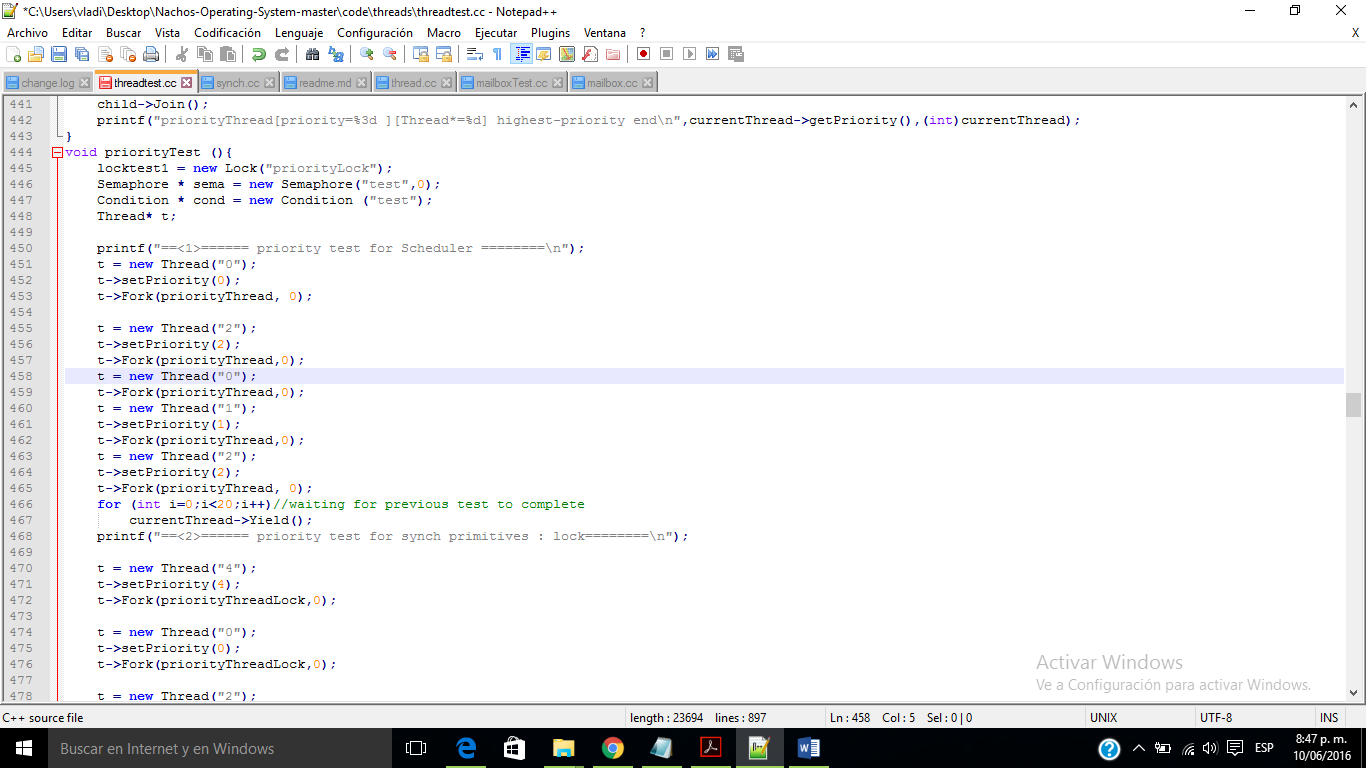
so that it exits before all mid-priority threads finish.

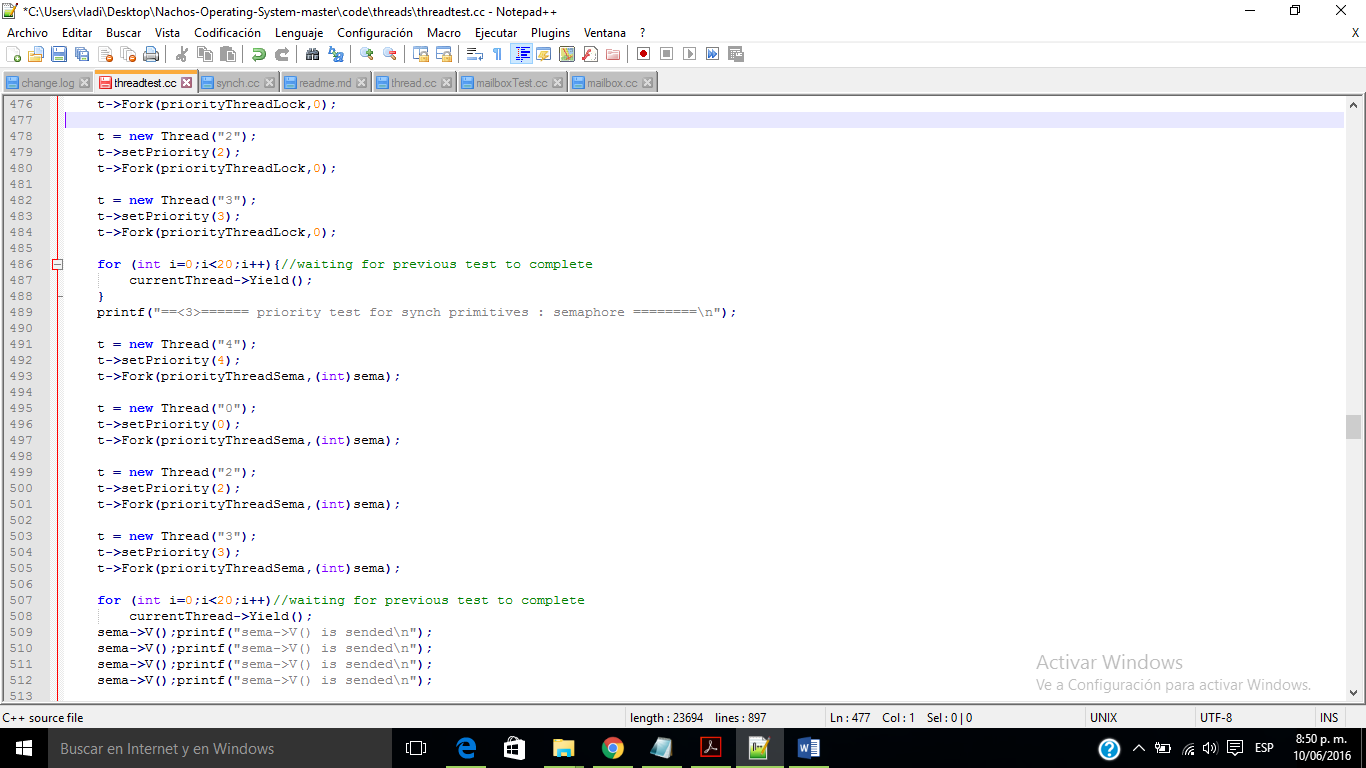
<7> [Extra Credit] There are 4 threads in this case. The joinee is successfully promoted, so that it

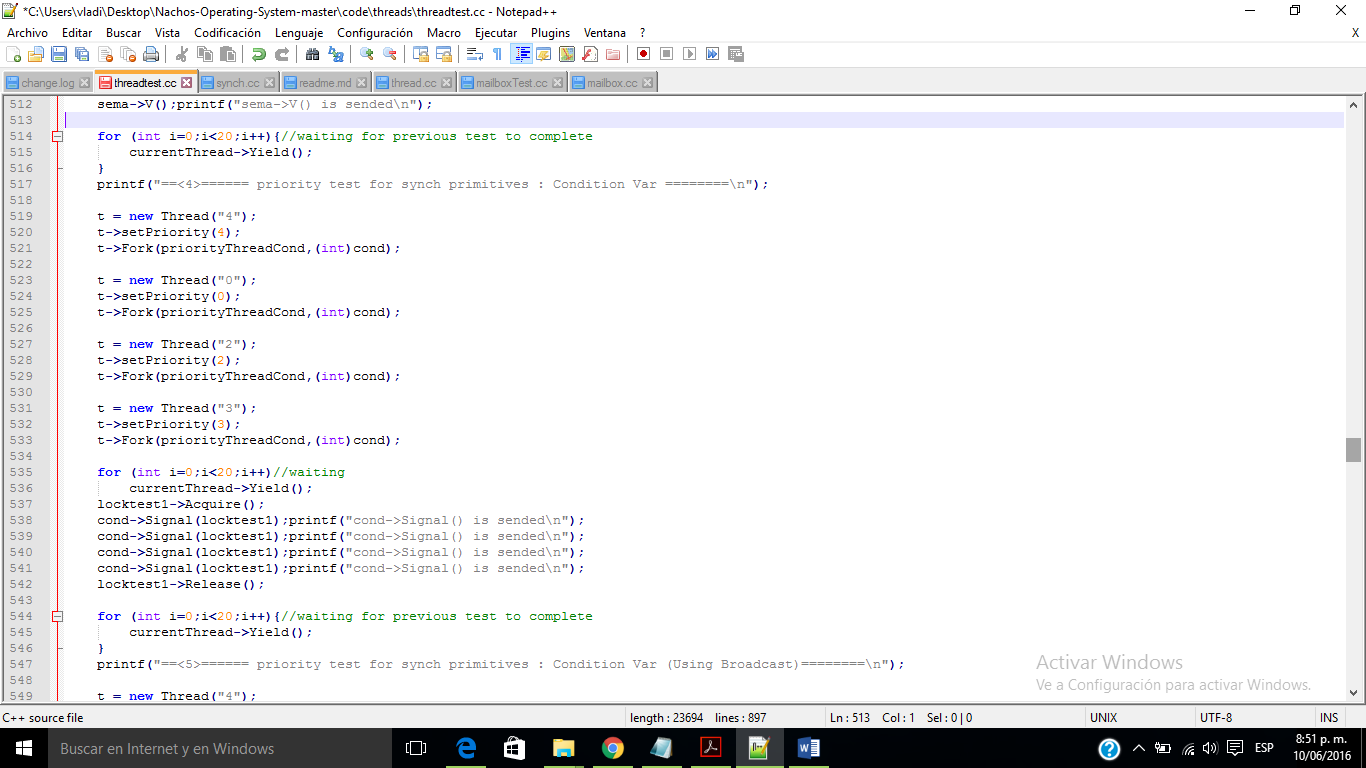
exits before all mid-priority threads finish.

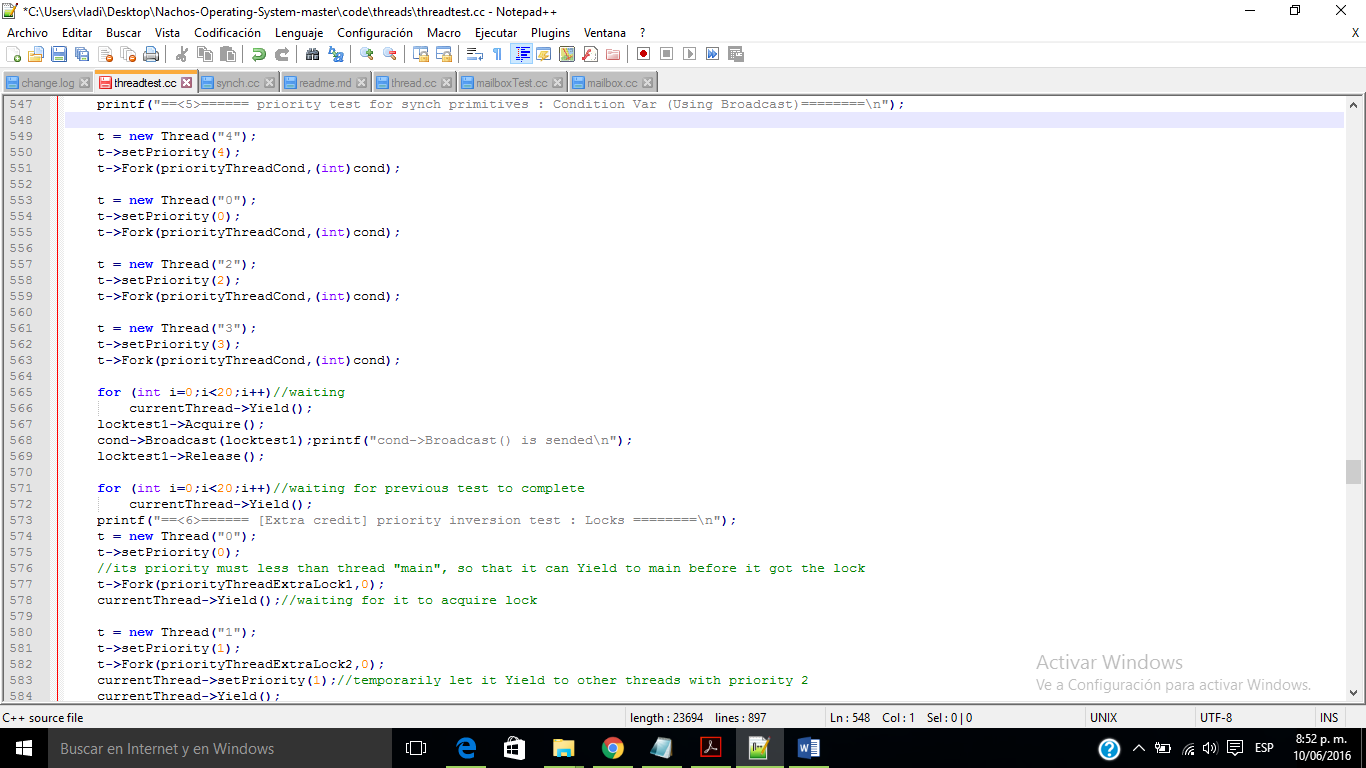
<8> [Extra Credit] There are 6 threads in this case. When Join() is called, the joinee (it is also the lock waiter) is promoted, as well as the lock holder. So that they exit before all mid-priority threads.

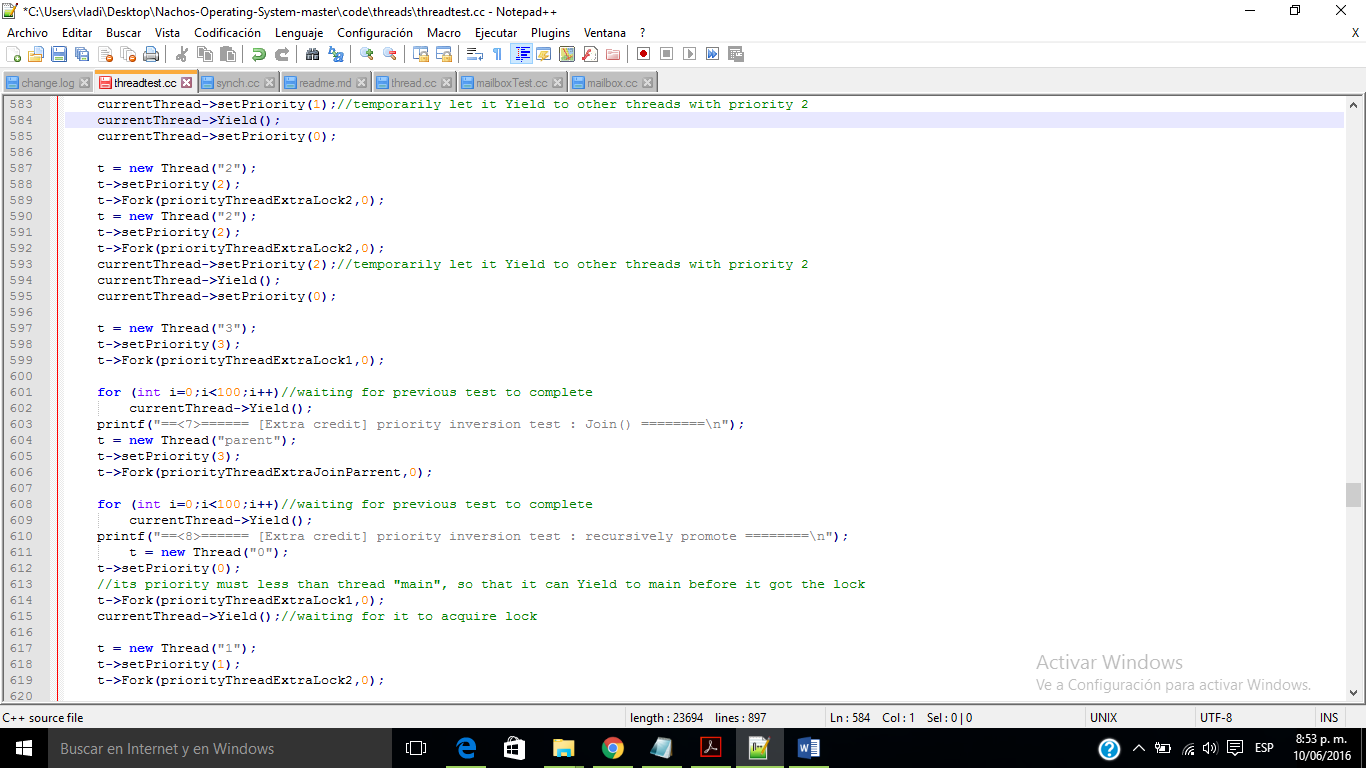












#Whales

###Whales

We used 4 semaphores to implement whale matching. There are 3 functions with similar codes male, female and matchmaker representing whales from each class. There are four semaphores thus four wait lists, one for each whale class to make sure only one whale from each class is waiting to be matched, and another for whales to wait for being matched. Only when 3 whales from each of the 3 classes come, can they form a match and then return, otherwise wait. To make things clear, we add variables to indicate the number of match and whales from each classes, when a male whale comes or returns, the program will print "one male whale come(return from match NO.#)" and then print out the number of male whales waiting to be matched, when female and matchmaker come or return, the program will print similar message. And when a match

completed it will print "match NO.# completed"

Test switch case numbers used:

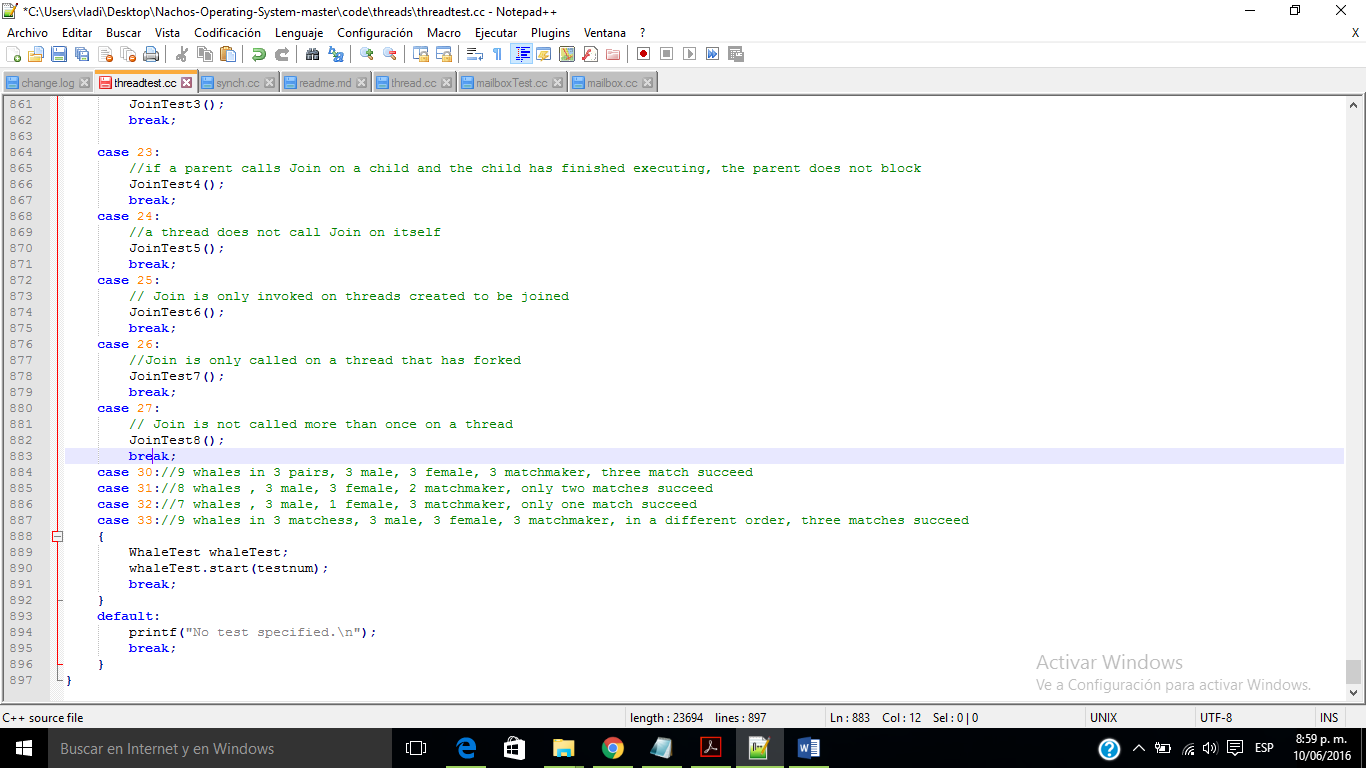
30: 9 whales in 3 pairs, 3 male, 3 female, 3 matchmaker, three matches succeed

31: 8 whales , 3 male, 3 female, 2 matchmaker, only two matches succeed

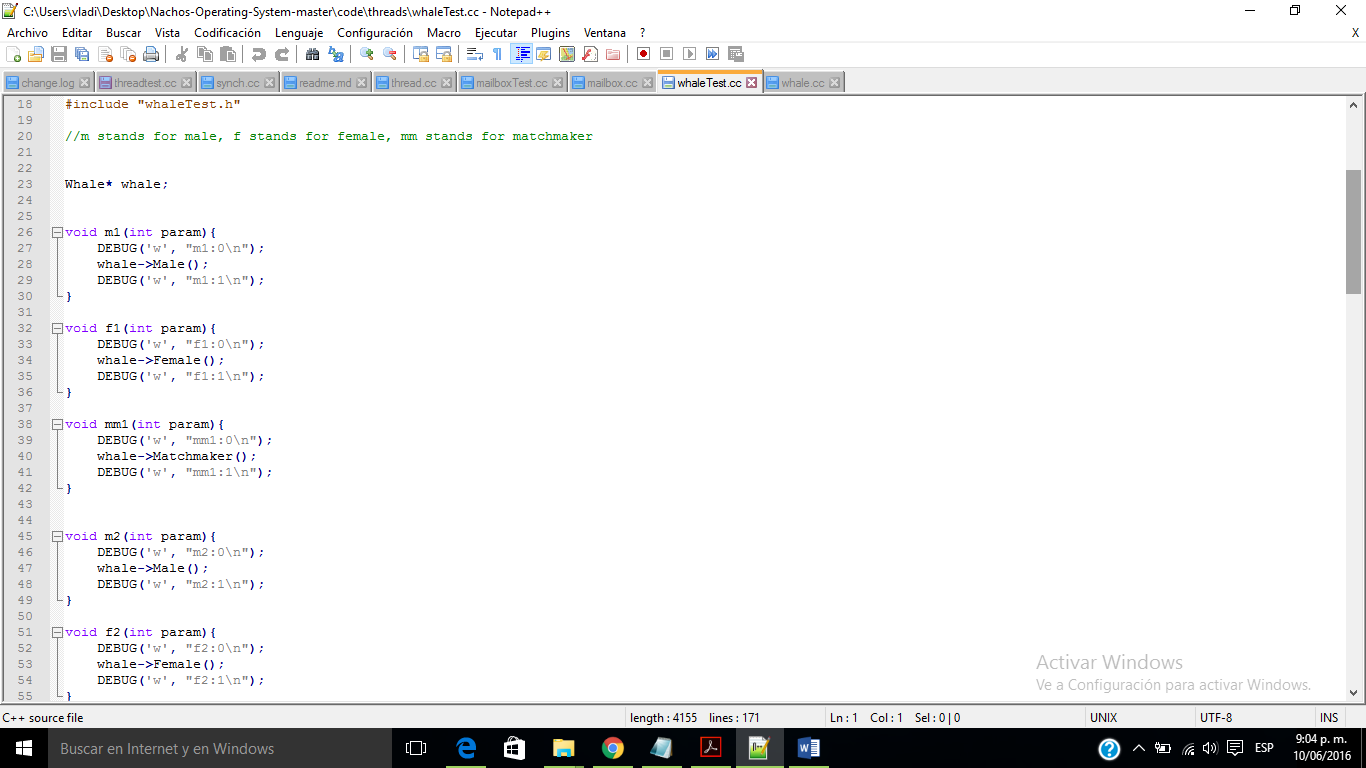
32: 7 whales , 3 male, 1 female, 3 matchmaker, only one match succeeds

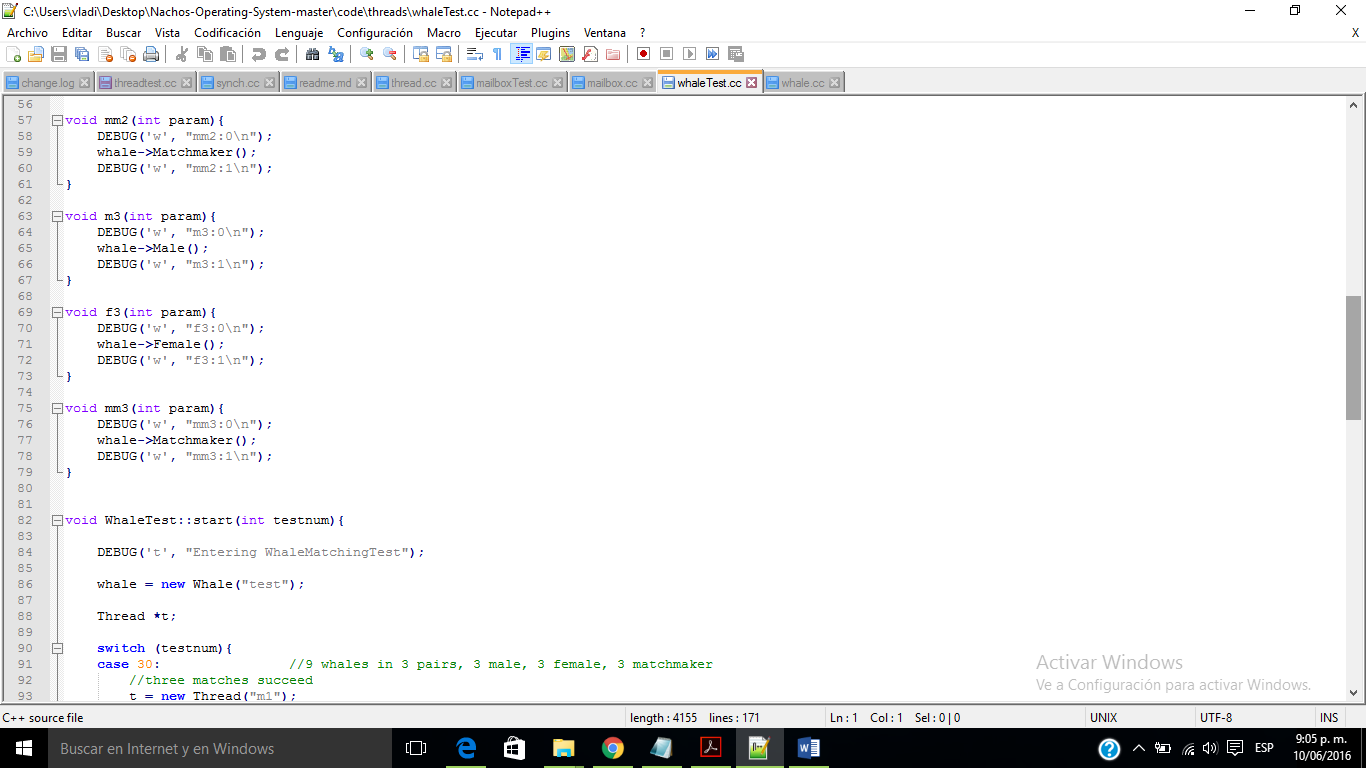
33: 9 whales in 3 pairs, 3 male, 3 female, 3 matchmaker, in a different order, three matches succeed we also tried -rs while testing and our program works fine in random switch circumstance

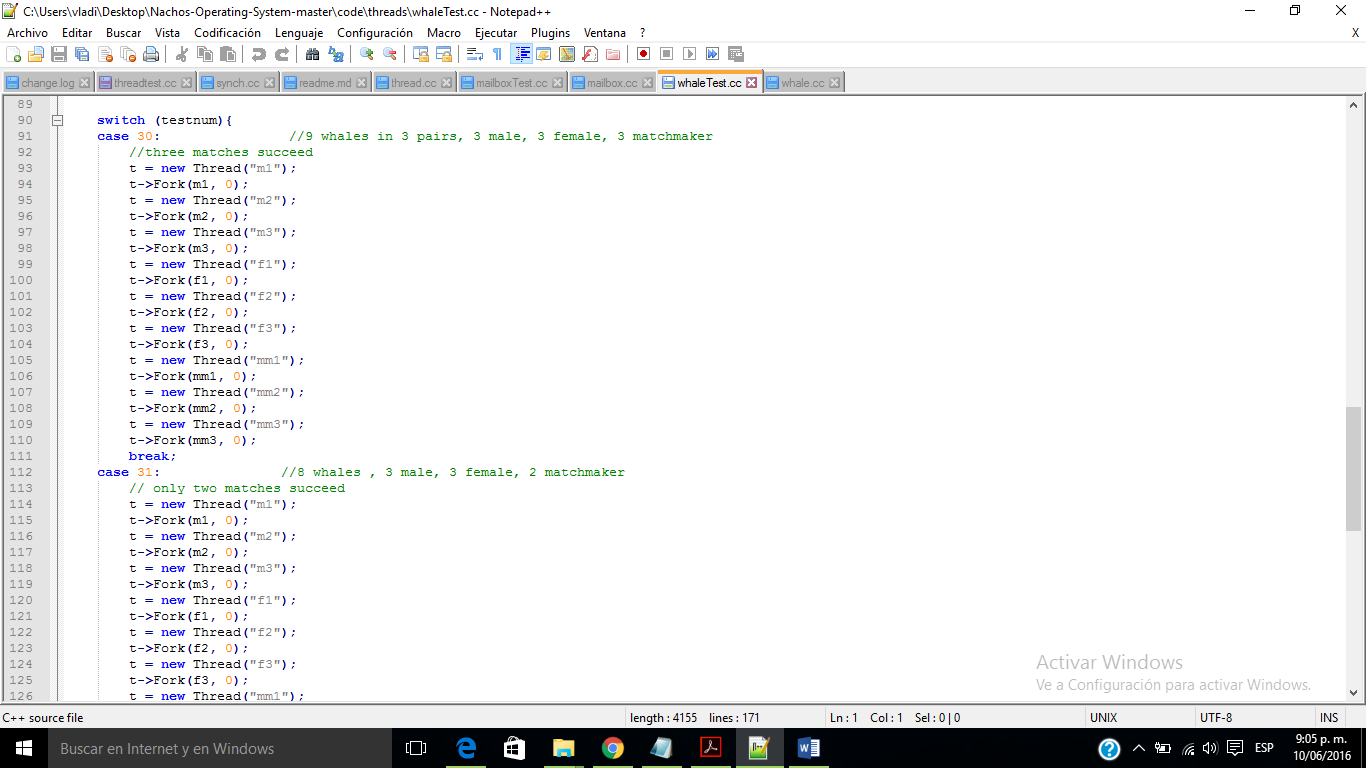
use -d w and -d x to see debug information

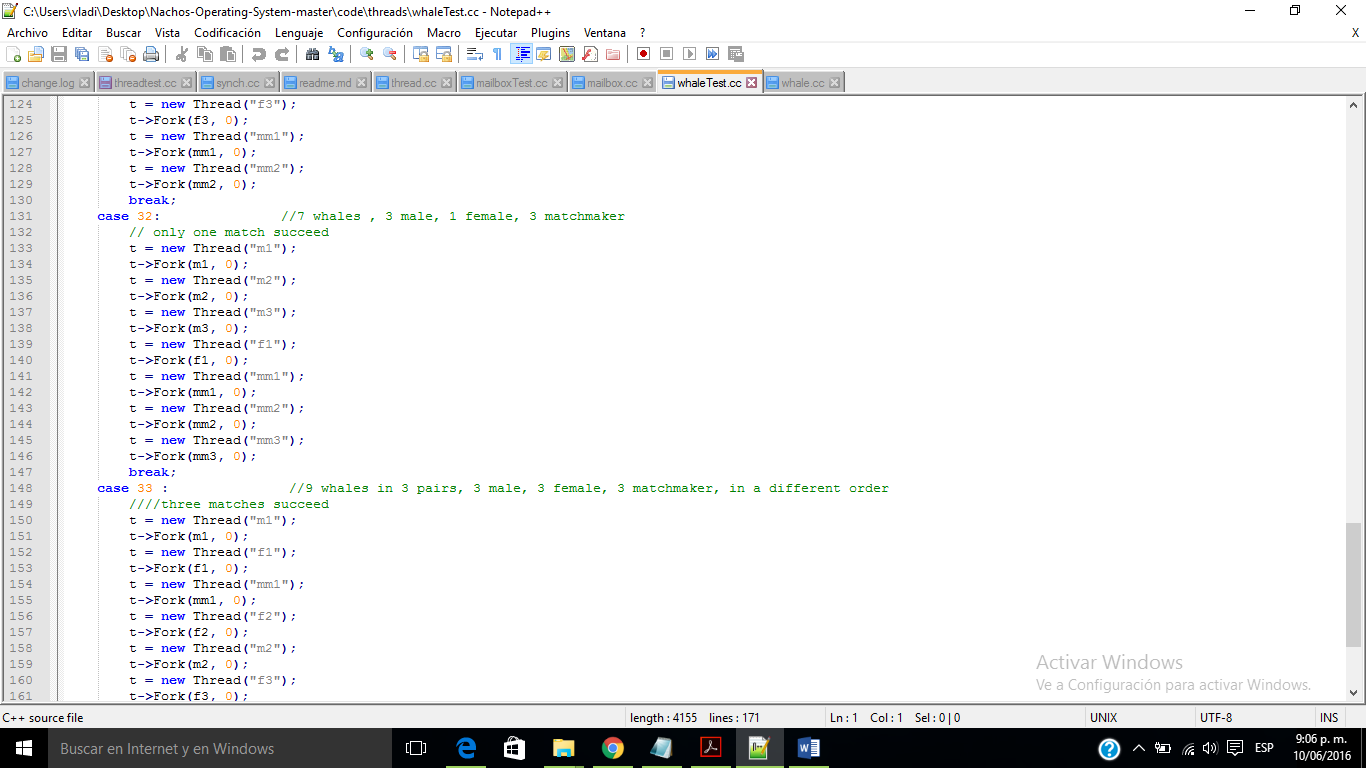


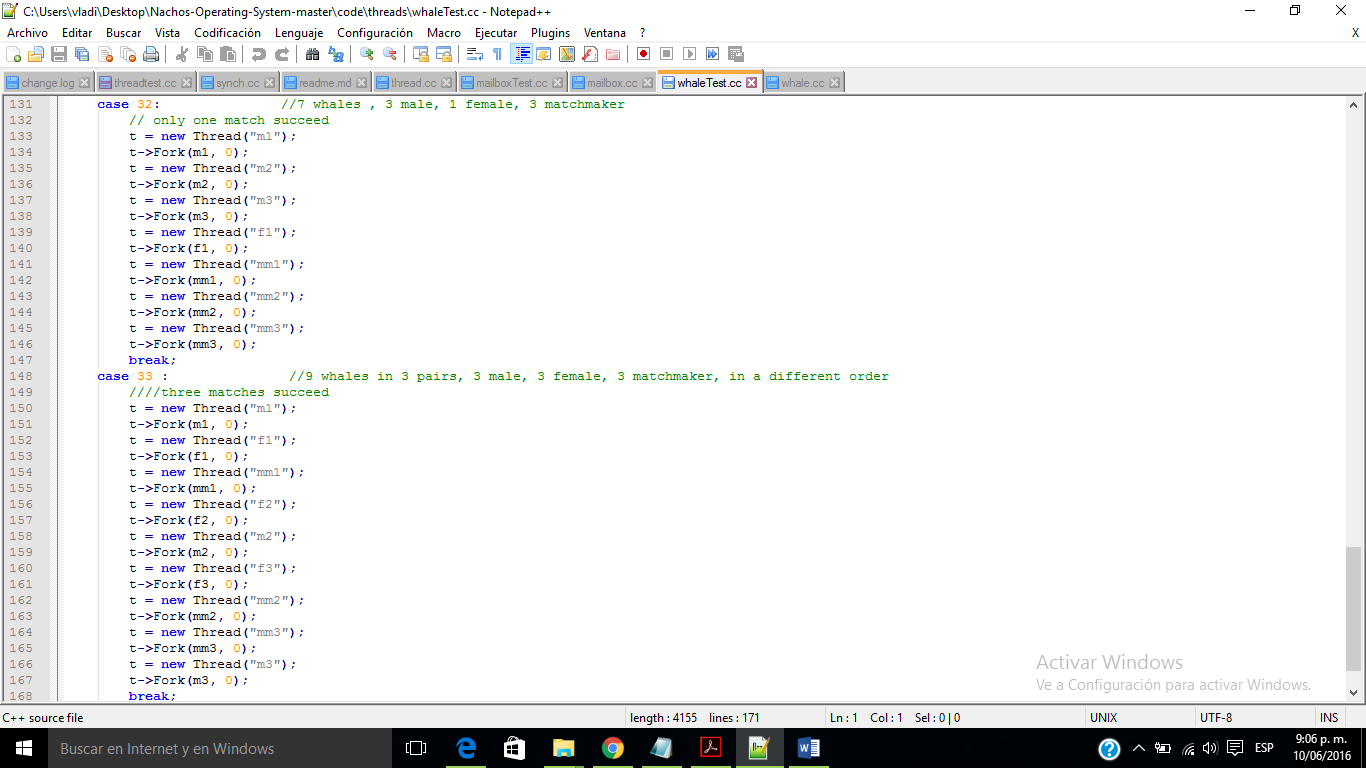
Whale test



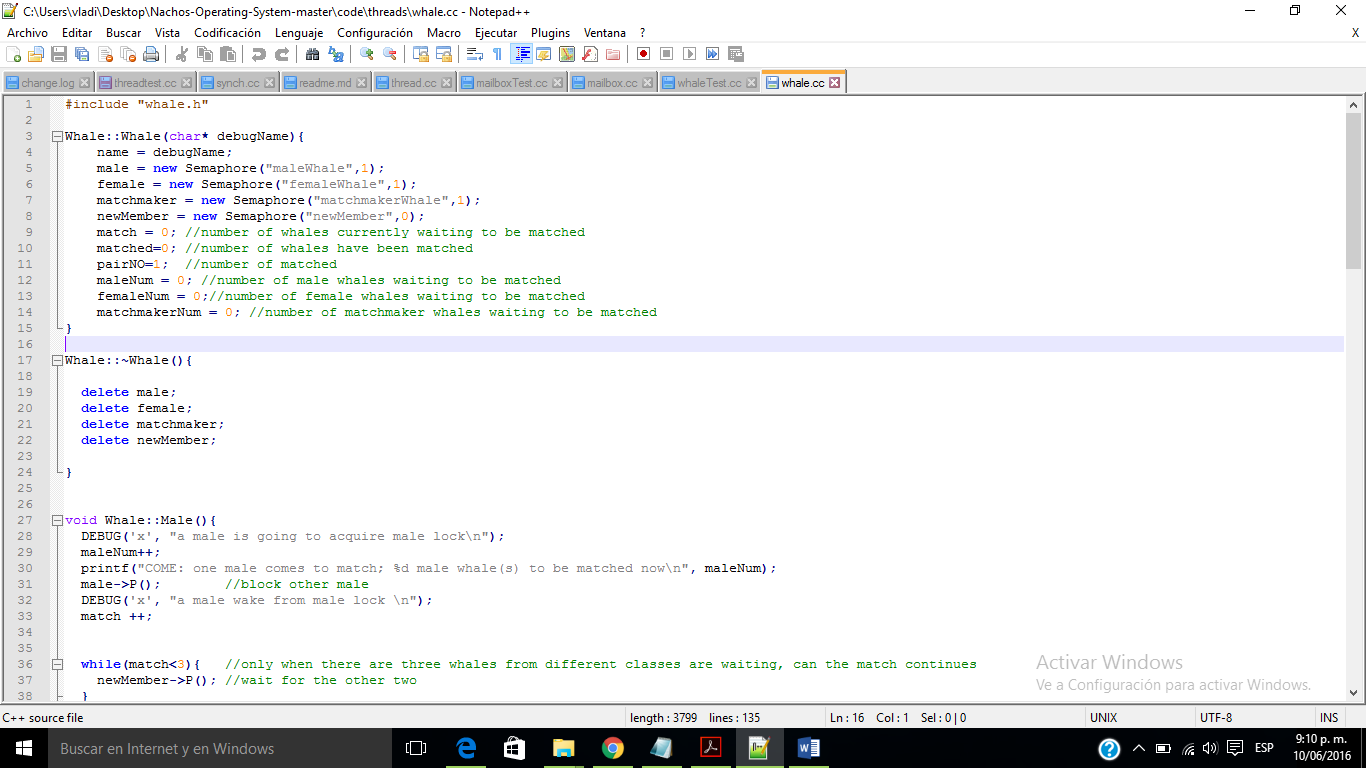


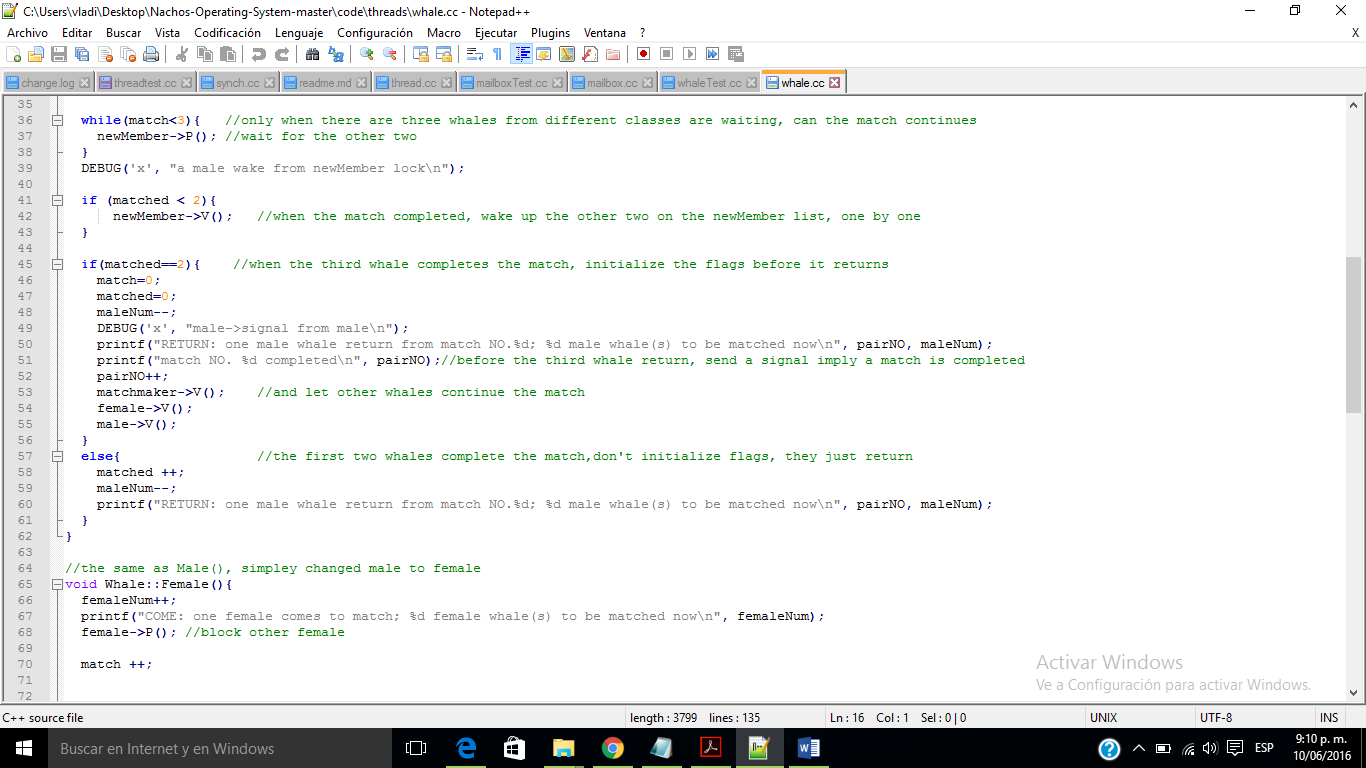


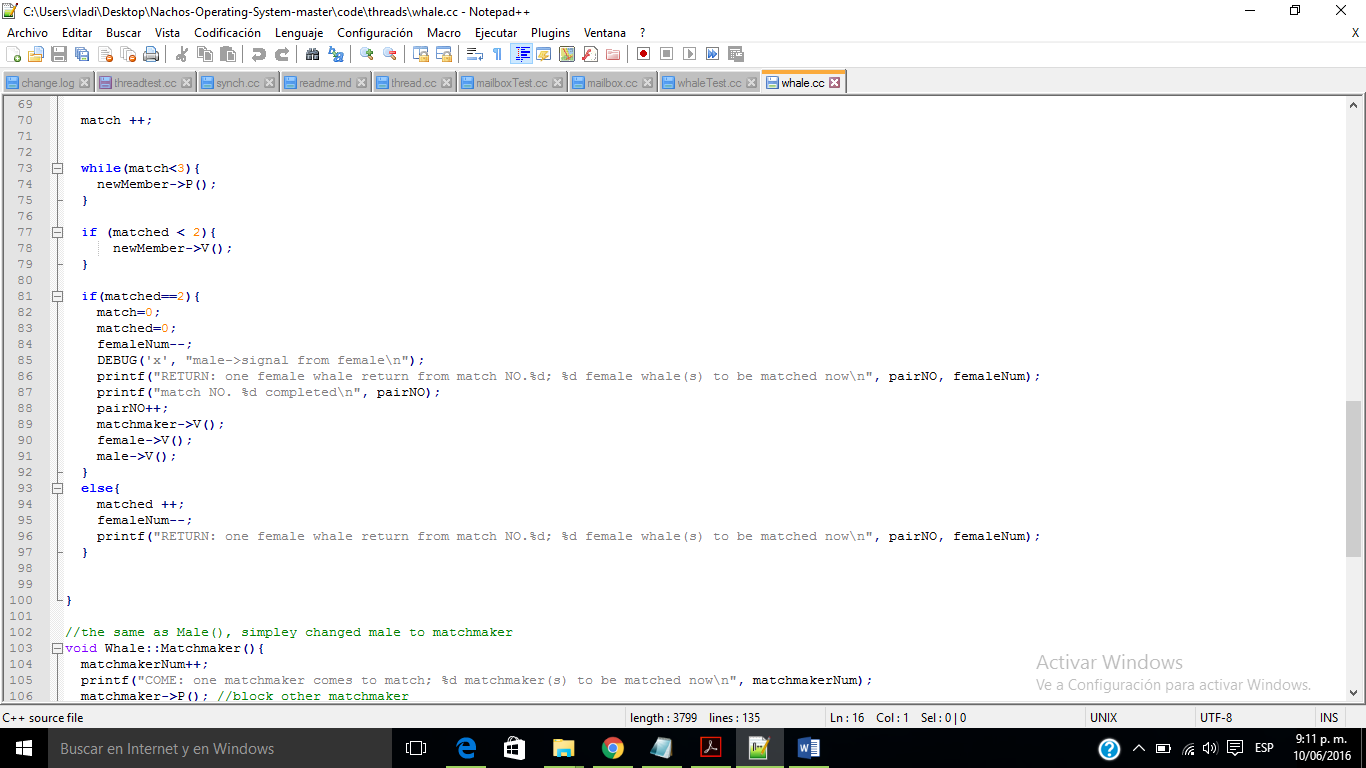


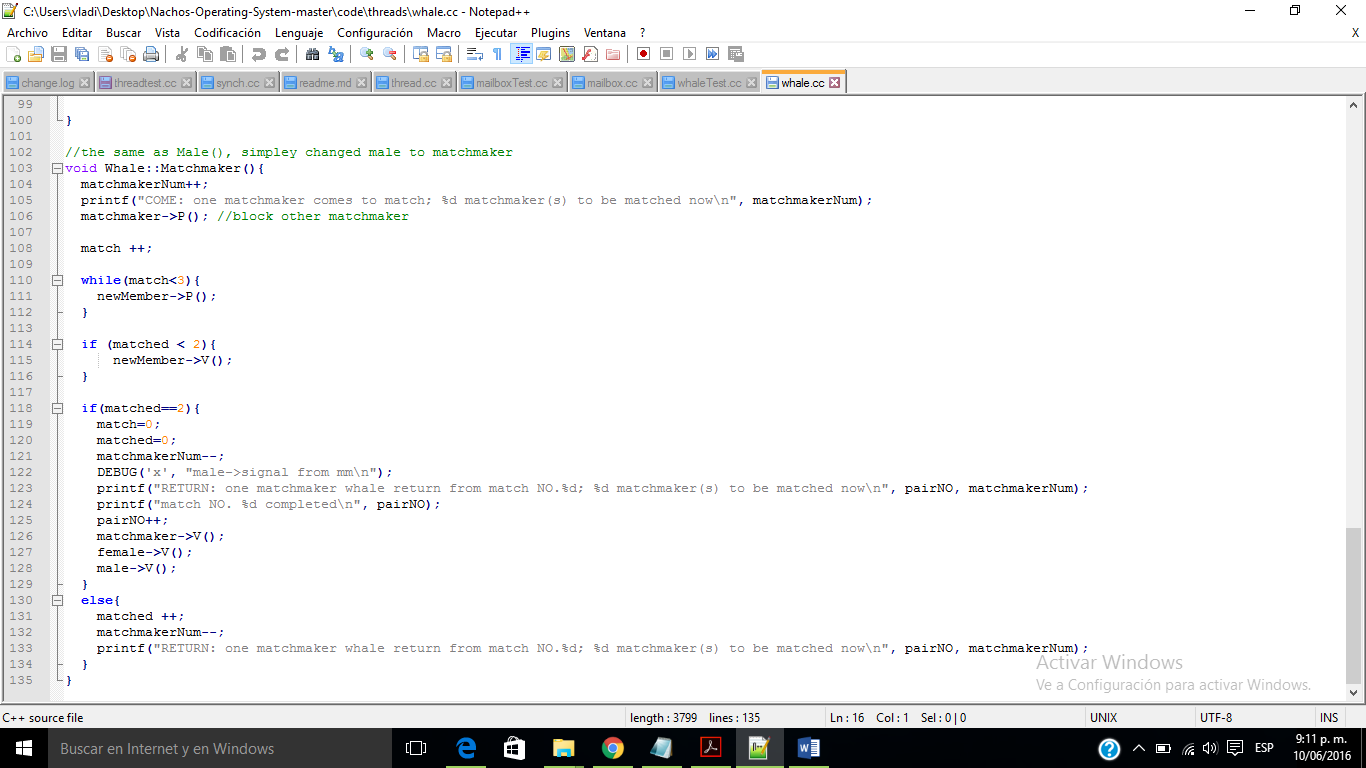


Whale









./nachos -q 33

