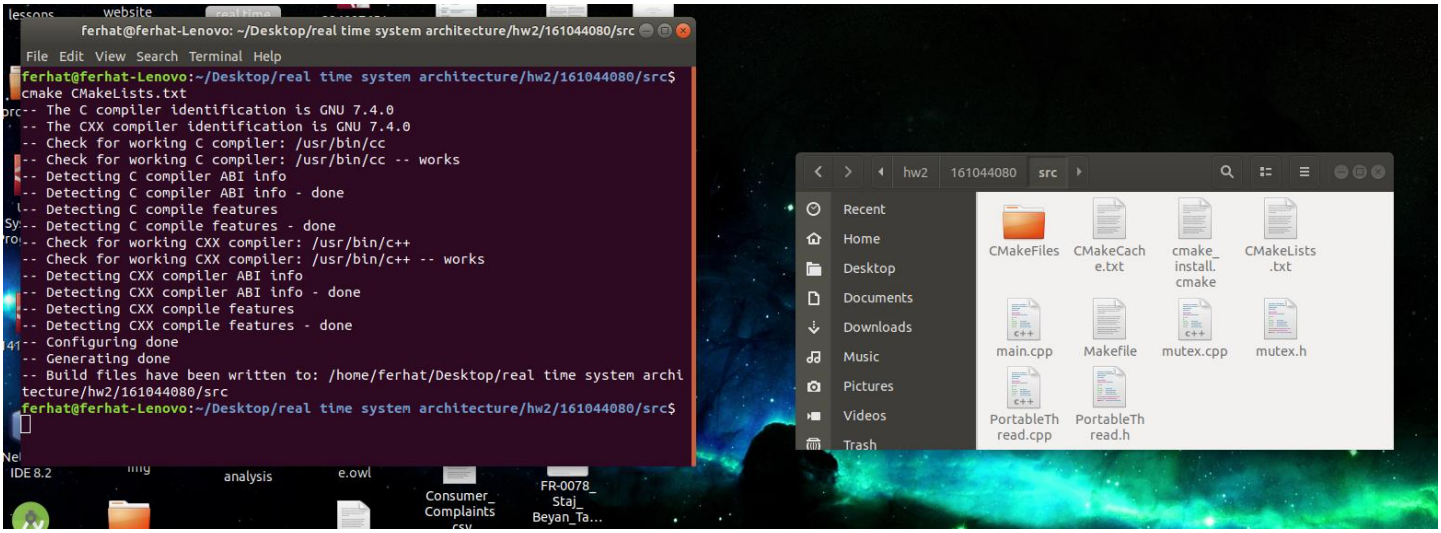
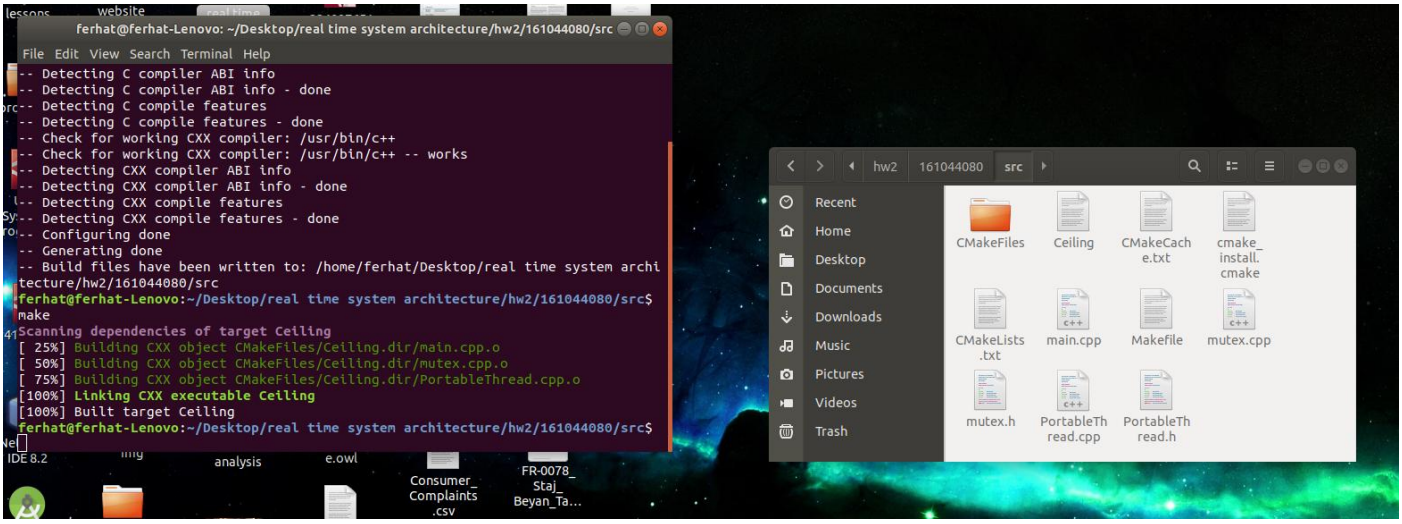
How to build and test project :

For Linux systems :

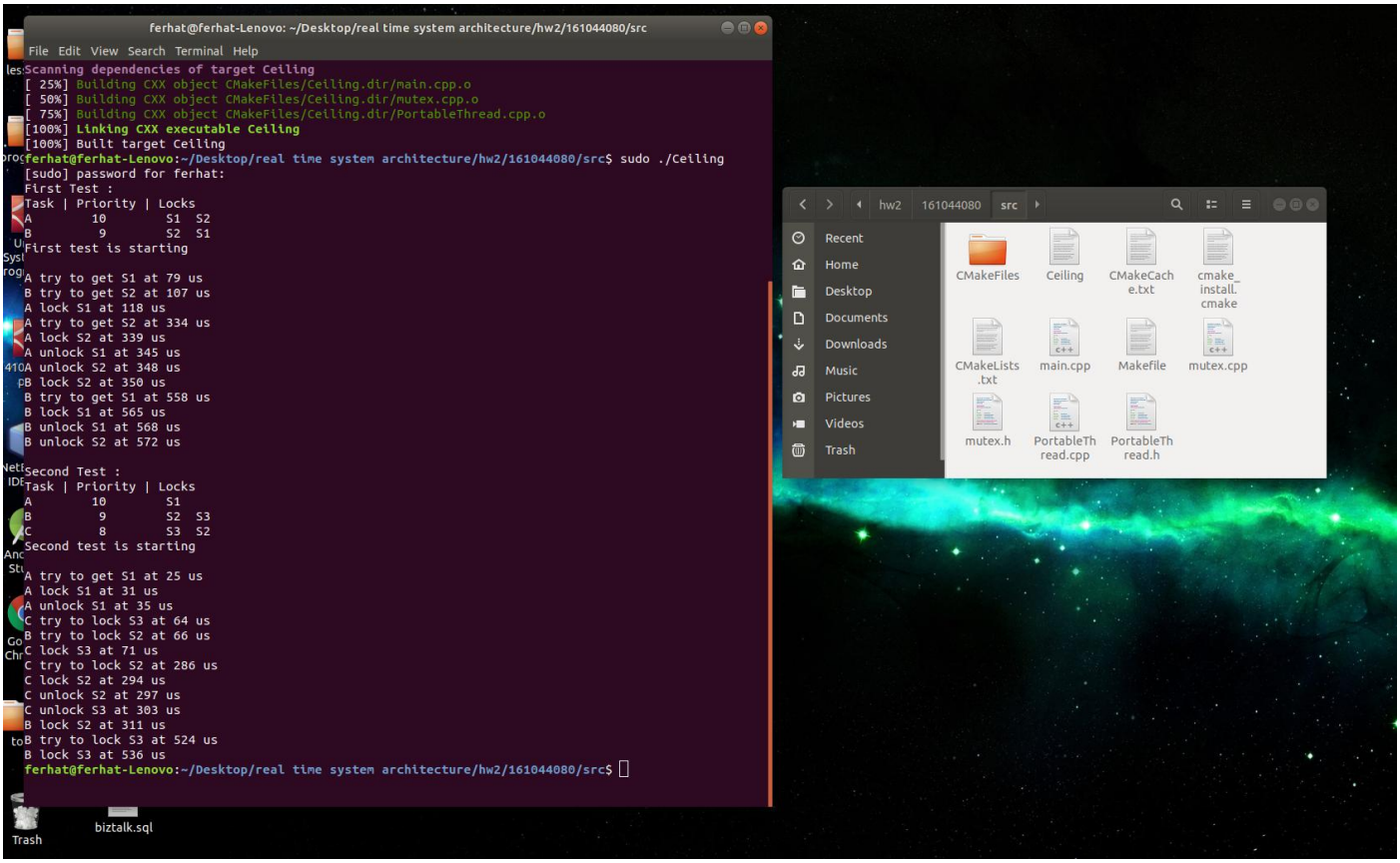


- Run “cmake CMakeLists.txt” command to create a makefile for the project.



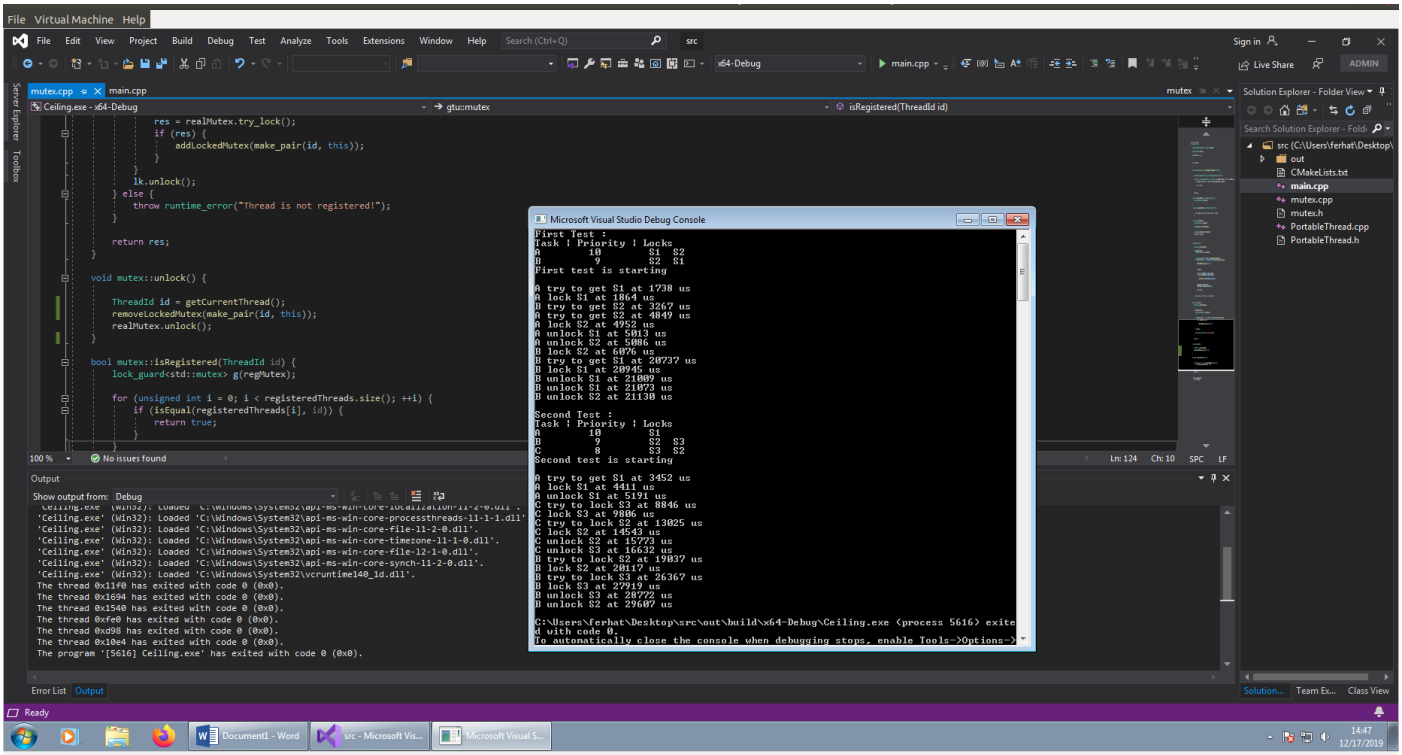
- Run “make” command to create executable files.

Note That : Program should be run as root user in Linux, otherwise system calls to change priority will not work.



- Run “sudo ./Ceiling” command and enter root password to run the program.

For Windows :



- Open project in Visual Studio as Cmake project and set main.cpp as startup item.

Design of project :

std::mutex is used inside gtu::mutex to handle lock and unlock. Thread that want to use mutex will register itself by calling the registerThread function and if their priority is bigger than the mutex ceiling, ceiling will be set to that priority. Thread ids are saved in a vector and if a thread uses the mutex, mutex will check the thread if it is saved or not by calling isRegistered method.

If a thread calls lock function to get lock, mutex will check if there are any other mutexes locked by other threads to compare the priority of the thread with the ceiling of mutex. If priority is not bigger than ceiling of other locked mutexes than thread will wait for the highest mutex to unlock. If Priority of thread is bigger than the thread that holding locked mutex then priority inheritance will happen and priority of the thread that holds the lock will be elevated until it unlocks the mutex. If priority of thread is bigger than the ceiling of other mutexes then thread locks the mutex.

When a thread gets the lock, it is saved to the lockedMutexes list for other mutex to see that it got locked. This way, other mutex can call getHighestCeilingMutex function to get highest ceiling mutex and the thread that holds it.When thread unlocks the mutex, it is removed from the lockedMutexes list.

Gtu::mutex implements the std::mutex so that it can be used by unique_lock and lock_guard class.

2 different test is used to show that program is running.

In the first one, There are 2 threads A and B. A lock S1 and S2 anb B locks S2 and S1.

A lock(S1) sleep(200us) lock(S2) unlock(S2) unlock(S1)

B lock(S2) sleep(200us) lock(S1) unlock(S1) unlock(S2)

Sleep is used to force thread to context switch between each other to show that deadlock is not created.

In the Second test, There are 3 threads A, B and C.

A lock(S1) unlock(S1)

B lock(S2) sleep(200us) lock(S3) unlock(S3) unlock(S2)

C lock(S3) sleep(200us) lock(S2) unlock(S2) unlock(S3)