CSE 437

REAL TIME SYSTEM ARCHITECTURES

HOMEWORK 2 REPORT

Ahmet Yuşa Telli

151044092

In main function, we have 5 different threads and 3 threads are use thread_func_A and 2 threads are use thread_func_B. They have different priority. And we have two mutexes. In these thread functions, first we lock (unique lock) our condition variable. Then, these functions lock (lock guard) mutexes. A function first lock mutex s1 then lock mutex s2. B function first lock mutex s2 then lock mutex s1. In these functions, we take scheduling policy and parameters of the thread.

After take parameters we need to set scheduling parameters. Then, register these threads to our mutex class.

We create "*gtu namespace*". In this namespace, we have two classes. First class is "thread_save_class". In this class, we save our threads' information; ID, priority, mutex vector.

Second class is "mutex" class. This class inherit std library mutex. We apply *priority ceiling protocol* in this class. In private part, we save ceiling number, acquire, own condition variable, own mutex (std) and first class' vector.

First, we register all threads in the vector. Thread comes from parameter and take it's ID, priority and push to our vector then set ceiling with check priority. Before set ceiling, we need to check it's priority. If it is maximum priority, we set it is ceiling number.

When finding to maximum priority, we check our thread's vector and return the maximum priority.

Lock:

When we lock, first we need to check is thread saved before. Because if thread wants to use mutex, it has to be registered before. If registered before, then check mutex is enable. This meaning is check mutexes acquires. If a mutex is free, then the thread lock this mutex and the mutex's acquire is true. But, if the thread's priority is greater than the others ceiling numbers, then the thread lock this mutex. If less than ceiling number, the thread needs to wait with condition variable.

Unlock:

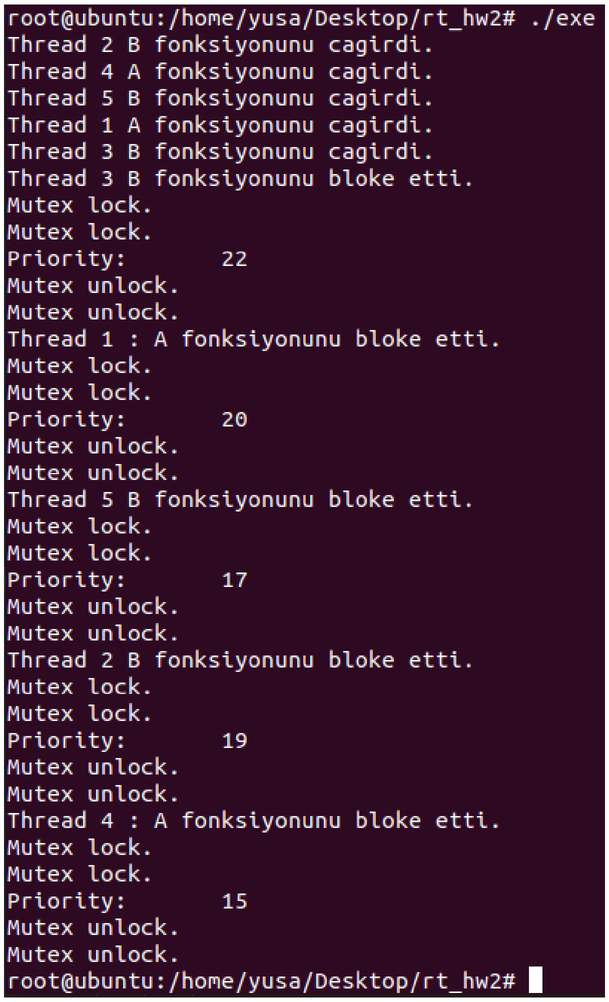
For unlock the mutex, we remove the thread in our thread's vector. After we update the ceiling number and notify one sleep thread. Then the mutex's acquire needs to be false, for others to use.

For all of these, we have 2 vectors, one for mutexes one for threads. We check mutex acquires and use condition variable and update ceiling.

Running:

Unix:

```
$ cmake .
$ make
$ sudo su
$ ./exe
```



```
root@ubuntu:/home/yusa/Desktop/rt_hw2# ./exe
Thread 2 B fonksiyonunu cagirdi.
Thread 4 A fonksiyonunu cagirdi.
Thread 5 B fonksiyonunu cagirdi.
Thread 1 A fonksiyonunu cagirdi.
Thread 3 B fonksiyonunu cagirdi.
Thread 3 B fonksiyonunu bloke etti.
Mutex lock.
Mutex lock.
Priority:       22
Mutex unlock.
Mutex unlock.
Thread 1 : A fonksiyonunu bloke etti.
Mutex lock.
Mutex lock.
Priority:       20
Mutex unlock.
Mutex unlock.
Thread 5 B fonksiyonunu bloke etti.
Mutex lock.
Mutex lock.
Priority:       17
Mutex unlock.
Mutex unlock.
Thread 2 B fonksiyonunu bloke etti.
Mutex lock.
Mutex lock.
Priority:       19
Mutex unlock.
Mutex unlock.
Thread 4 : A fonksiyonunu bloke etti.
Mutex lock.
Mutex lock.
Priority:       15
Mutex unlock.
Mutex unlock.
root@ubuntu:/home/yusa/Desktop/rt_hw2#
```

Windows :

We need to install Cmake for windows. Then, select project file and “Configure” and “Generate” our project. Created make files. I select *build* in project file for build binary files.

We use Visual Studio 2017, VS run as administrator. Open the .sln file in project’s build file. On “Solution Explorer” find “exe” then right click, select “Set as StartUp Project”. On the “Debug” menu, “Start Without Debugging” select and run the project.

Not Completed.