CSE 437
REAL TIME SYSTEM ARCHITECTURES

HOMEWORK 02 REPORT

Ömer ÇEVİK

161044004

In this part, creating a mutex concrete class in gtu namespace which extends *std::mutex* concrete class. The mutex concrete class has *lock()* and *unlock()* methods to override *std::mutex* concrete class this methods.

In gtu namespace, created a new concrete class which is called $RegThreadInfo$. That class used to register the created threads by users to created mutexes. The *gtu::mutex* concrete class keeps in private as an vector/array of $RegThreadInfo$ object informations as registered thread's priority, id and vector/array of registered mutexes.

In main function, creating 20 threads as an example which uses *thread_1_function()* an *thread_2_function()* functions that doesn't have any parameter. Created threads passed to *gtu::setScheduleThread()* giving first parameter as thread and second parameter as thread's priority. The *gtu::setScheduleThread()* function sets the thread's priority for linux and windows correctly and registers the thread to each mutexes, used 3 mutexes in that example. After scheduling *gtu::startProtocol()* function must be called to start the priority ceiling protocol using conditional variable.

In *thread_1_function()* and *thread_2_function()*, locking mechanism is created using *lock_guard* in one of them and *unique_lock* in the other one for 3 mutexes. This functions use conditional variables to wait till register operation completed. This functions just increases a global integer variable. The *lock_guard* and *unique_lock* mechanism uses mutex's *lock()* and *unlock()* methods. The priority ceiling protocol is decided to be applied in these methods. In *lock()* method firstly checking the the current thread if it is registered to mutex if it is not has been registered yet then printing the exception message and not locking.

Otherwise checking the acquired mutex which threads have registered itself on it and the mutex has the current thread. Checking the current thread's priority comparison between acquired mutex's ceil and decide to lock or wait till acquire the mutex using conditional variable. If mutex has not been acquired then locking mechanism works and acquired re-initialized to true.

In this programme, printing to screen locked threads and unlocked threads between which function working on the thread. For an example created a thread which is not registered to any mutex. That causes an exception and prints to screen that thread's id.

Consequently, the deadlock problem is correctly solved using threads and mutexes with priority ceiling protocol for Windows 10 and Linux operating systems.

To run the programme in Linux operating system, using the *cmake* command on "161044004" directory and after that switch to root mode using *sudo su* command and enter make command. After *make* command enter the *./main* command on console.

To run the programme in Windows 10 operating system, after *cmake* operation for Visual Studio in "161044004" directory open the *161044004.sln* file in Visual Studio as administrator mode. Then make the project as startup project using *"set as startup project"*. Then to see the results on the screen *"Start Without Debugging"*.