Homework - 2 CSE437
REAL TIME SYSTEM ARCHITECTURES
Hamza YOĞURTCUOĞLU - 171044086

main.cpp : I created threads and called to setOSPriority function that is setting priority of threads according to operating system. Then main thread waits to created threads.

gtu.hpp : I handled gtu::mutex in this file. I implement priority celling protocol in **gtu** namespace.

How Designed Priority Celling Protocol System :

Priority ceiling protocol is a synchronization protocol for shared resources to avoid unbounded priority inversion and mutual deadlock due to wrong nesting of critical sections.

```
void fourDecreament(){

    m1.registerThread();
    m2.registerThread();
    for (size_t i = 0; i < 10; i++)
    {
        std::lock_guard<gtu::mutex> lock1(m1);
        std::lock_guard<gtu::mutex> lock2(m2);
        resourceValue=resourceValue-4;
        cout<<"fourDecreament : "<<resourceValue<<endl;
    }
}

void threeDecreament(){

    m1.registerThread();
    m2.registerThread();
    for (size_t i = 0; i < 10; i++)
    {
        std::lock_guard<gtu::mutex> lock1(m2);
        std::lock_guard<gtu::mutex> lock2(m1);
        resourceValue=resourceValue-3;
        cout<<"threeDecreament : "<<resourceValue<<endl;
    }
}
```

Thread functions has common resource that is resourceValue. I lock m1 and m2 mutexes but other threads try to lock m2 and m1 mutexes. But it is not deadlock through gtu::mutex.

I handled communication of mutexes with mutex array. Because if a thread is registered m1 and m2 and the thread has 10 priority but other thread has 11 priority. If 11 priority thread locks m1. 10 priority thread cannot lock m2 mutex. So,that must communicate with among them. Each mutex must celling priority 11. Consequently , other thread priority is not bigger celling of mutex.

```
void mutex::lock(){

    try
    {
        std::this_thread::sleep_for(std::chrono::milliseconds(20));
        registeredOrNOT();
        std::unique_lock<std::mutex> lk(m);
        cv.wait(lk, [&]{
            return checkGetIn();
        });
    } catch(const std::exception& e)
    {
        std::cerr << e.what() << '\n';
    }
    id = std::this_thread::get_id();
}
```

Firstly I'm checking either thread register or not to mutex. If thread is not registered. I throw exception. Then I controled **mutexArray[i]→current>sch.sched_priority.** If there is any thread has locked mutex. Current will be celling priority or -1. the priority of the thread holding the lock will be elevated if necessary.

handledOS.hpp : I set priority with setOSPriority function.

Linux :

For Linux can be 1-99 priority. You must be root user. If you use this function other way that you will get exception.

```
#if defined(__linux__)
    sched_param sch_params;
    sch_params.sched_priority = priority;
    if(pthread_setschedparam(th.native_handle(), policy, &sch_params)) {
        std::cerr << "Failed to set Thread scheduling : " << std::strerror(errno) << std::endl;
    }
```

Windows: I categorized 6 parts of priority sections.

1) Thread_priority_time_critical
2) Thread_priority_highest
3) Thread_priority_above_normal
4) Thread_priority_normal
5) Thread_priority_below_normal
6) Thread_priority_lowest

So, I can handle priority celling protocol with OS interface.

Test : I have 8 threads that are 4,3,2,1 decreasing and increasing. They have priority. So, they work according to priorities. Resource is started with 0 and finished with 0.

Start → 0

```
fourDecreament : -4
fourIncrement : 0
fourDecreament : -4
oneDecreament : -5
twoIncrement : -3
fourIncrement : 1
```

End → 0

```
oneDecreament : -5
oneIncrement : -4
twoIncrement : -2
oneIncrement : -1
oneIncrement : 0
```

Compiler and run :

firstly : → cmake . #cmake command if you are same direction '.' or you must give absolute path

secondly : → make #make command is create object code that means compile with g++.

Test : Just create your thread and set priority.

```
std::thread t1(oneIncreament);
setOSPriority(t1, SCHED_FIFO, 12);
t1.join();
```

Note: You must register in your function which mutex you will use. If you forget it hasn't registered itself before, a runtime exception will be raised.

```
m1.registerThread();
```