CSE 437 REAL TIME SYSTEM ARCHITECTURES
HOMEWORK 2 REPORT

- **The requirements of your timer, containing constraints and assumptions.**
  - Öncelikle **gtu::GtuMutex** class'ını yazmak için C++11 foksiyonelliğini ve bazı özelliklerini kullandım.

  - Bu GtuMutex class'ını implement etmek için **Helper** adında bir yardımcı data class'ı yazmaya ihtiyacım oldu. Bu class içinde ise **Priority Ceiling Protocol'ü** implement edebilmek için gerekli gördüğüm değişkenler vardır. Bu değişkenler şu şekildedir:

```
class Helper
{
public:
    std::thread::id worker_thread;
    int priority;
    bool isBlocking;
};
```

  - Yazmış olduğum bu sistemin hem **Windows** hem de **Posix** sistemlerde çalışabilmesi için platform bağımlı kod yazmam gerekti bunun içinde aşağıdaki tanımlamaları yapmam gerekti.

```
#ifdef _WIN32
#include <Windows.h>
#define WINDOWS
#elif defined(_WIN64)
#include <Windows.h>
#define WINDOWS
#elif defined(unix) || defined(__unix__) || defined(__unix) || defined(__linux)
#include <unistd.h>
#define LINUX
#endif
```

- Son olarak GtuMutex class'ını implement etmem için **lock() ve unlock()** fonksiyonları dışında private olacak şekilde ekstra fonksiyonlar ve bir tane **register fonksiyonu** yazmam gerekti. Bu fonksiyonlar aşağıdaki gibidir:

```
class GtuMutex
{
private:
    std::vector<gtu::Helper> workers;
    volatile unsigned long lockVariable;

    int getIndexOfThread(std::thread::id thread_id);
    void setIndexOfThread(int index, int priorty);
    bool isThreadBlocking(int index);
    int getIndexOfBlockedThread();
    void setThreadStatus(int index);
    bool getThreadStatus(int index);

public:
    GtuMutex();
    ~GtuMutex();
    void registerMutex(std::thread::id thread_id, int prority);
    void lock();
    void unlock();
};
```

- **The design of the priority ceiling protocol system**
  - Öncelikle bu sistemi implement etmek için mutex'e register olacak thread'ın id'sini ve priority'sini alan bir tane **registerMutex** adında fonksiyon yazmam gerekti. Bu fonksiyon ile id'si verilmiş olan thread bu mutex üzerinde lock ve unlock olup olamayacağı kontrol edilebilecek. Ek olarak bu fonksiyon ile register olan threadin status'ü register olduğu anda porotokol kurallarına göre set edilecektir. Bu fonksiyonun implementasyonu bu şekildedir:

```
void GtuMutex::registerMutex(std::thread::id thread_id, int prority)
{
    Helper helper;
    helper.worker_thread = thread_id;
    helper.priority = prority;
    helper.isBlocking = false;

    workers.push_back(helper);

    int index = getIndexOfThread(this_thread::get_id());
    setThreadStatus(index);
}
```

- Register olan thread artık register olduğu mutex üzerinde **lock_guard, lock_unique** gibi lock'lama sistemi ile lokc'lanıp unlock edilebilecek hale gelmektedir.
- **Priority Ceiling Protocol'ü** implement etmek için **GtuMutex** içersindeki lock fonksiyonunda da bazı kkontroller yapmam gerekti. Öncelikle lock edilmesi istenilen thread'in belirtilen mutex üzerinde register olup olunmadığı control edilir. Eğer register olmuş ise ve priority'si protocol'e göre register olan diğer threadlerden öncelikli ise gerekli işlemler yapılır. Ve thread'lerin gerekli olan thread üzerinde priority ve statu değişikliği yapılır. Yazmış olduğum lock ve unlock fonsiyonu aşağıdaki gibidir.

```
void GtuMutex::lock()
{
    int index = getIndexOfThread(this_thread::get_id());
    if (index != NOT_REGISTERED && getThreadStatus(index))
    {
        setThreadStatus(index);
#ifdef WINDOWS
        for (; this->lockVariable == ONE ||
               InterlockedCompareExchange(&this->lockVariable, 1, 0) == ONE;)
        {
        }
#endif
#ifdef LINUX
        for (; this->lockVariable == ONE ||
               __sync_lock_test_and_set(&this->lockVariable, 1) == ONE;)
        {
        }
#endif
    }
}

void GtuMutex::unlock()
{
    this->lockVariable == 0;
    UNLOCK_MY_MUTEX(this->lockVariable)
}
```

- Bunların dışında bu protokolü implement etmek için bazı yardımcı fonksiyonlara ihtiyacım oldu bunlar da thread priority ve statüsünü set ve get etmek için ve arama yapmak amaçlı yazmış olduğum yardımcı fonksiyonlardır. Bu fonksiyonlardan biri o an istekte bulunan threadin register olup olmadığına bakan ve eğer register olmuş ise index'ini dönen **getIndexOfThread'in** implementasyonu aşağıdaki gibidir.

```
int GtuMutex::getIndexOfThread(std::thread::id thread_id)
{
    bool found = false;
    gtu::Helper registered;
    for (int i = 0; i < workers.size(); i++)
    {
        if (workers[i].worker_thread == std::this_thread::get_id())
        {
            return i;
        }
    }

    return NOT_REGISTERED;
}
```

- **How to build and test the this project?**
  - Bu projeyi build etmek için öncelikle **CMakeList.txt** dosyasının olduğu klasöre gidip aşağıdaki komutları çalıştırmalısınız:
    - **$cmake .**
    - **$cmake –build .**
  - Build işlemi sonrasın **gtuMutex** adında çalıştırılabilir bir dosya oluşturulacaktır. Bu dosyayıda aşağıdaki komut ile çalıştırabilirsiniz:
    - **$./gtuMutex**
  - Sistemi kendi test fonksiyonlarınız ile test etmek için için iki seçeneğiniz vardır.
    - Ya main fonksiyonun olduğu dosyada değişiklik yapabilirsiniz.
    - Ya da yeni bir test dosyası yazıp onu **CMakeList.txt** dosyasında ilgili yere yazıp eski main fonksiyonun olduğu dosyanın ismini silebilirsiniz.