CSE 437 - REAL TIME SYSTEM ARCHITECTURES
HOMEWORK 2 REPORT
Furkan OZEV – 161044036

a. The requirements of your timer, containing constraints and assumptions:

Constraints:

Construct registerTimers which can be different timers . These timers has different version :

- registerTimer(timepoint, callback): run the callback once at time point tp.
- registerTimer(period, callback): run the callback periodically forever. The first call will be executed as soon as this callback is registered.
- registerTimer(timepoint,period,callback): run the callback periodically until time point tp. The first call will be executed as soon as this callback is registered.
- registerTimer(predicate,period, callback): run the callback periodically. Before calling the callback every time, call the predicate first to check if the termination criterion is satisfied. If the predicate returns false, stop calling the callback.
- There should be 1 timer thread and 1 main thread
- If there is more than 10ms delay in the period, the deadline will print a miss warning.

Versions:

- Source code runs in C++ 11 version.
- The source code’s cmake runs in version 3.7

b. The design of the timer:

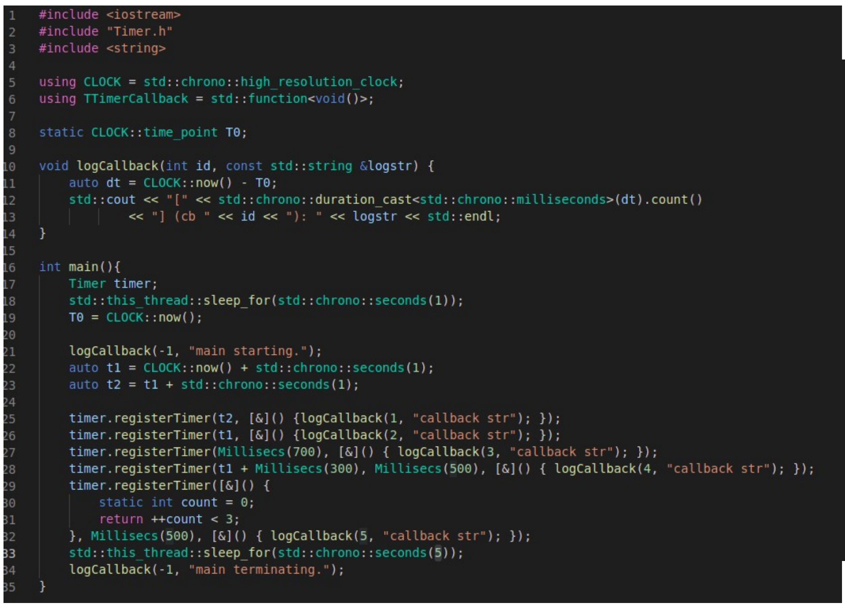
A class named TimerEntity was written to provide the information of the timers created in the RegisterTimer method. Timer was implemented which extends from ITimer.hpp. Inside the Timer class was an STL Vector holding the TimerTask objects. When we create timer object. We create a threat in order to handle timers. Then main thread is calling registerTimers. Then it notify the thread of timer. We are looking minimum remaining time. Then we put to sleep the class timer. When minimum remaining time comes. Then thread of class wakes up. But main thread must notify thread of class. We have just 2 threads which are class thread and main thread.

c. How to build and test your project:

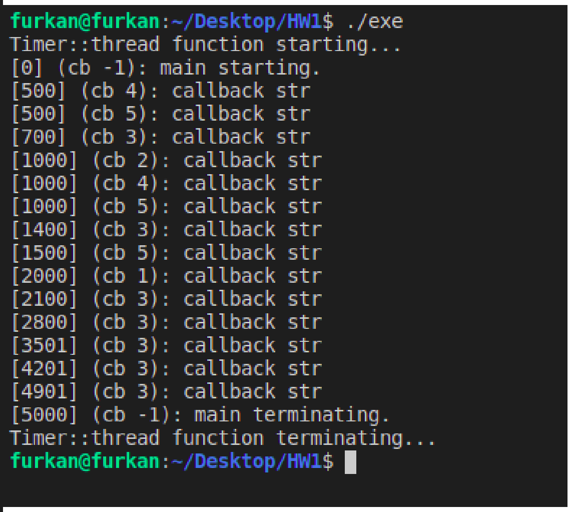
Compiler and run :

- → **cmake .** *# cmake command if you are same direction '.' or you must give absolute path*
- → **make** *# make command is create object code that means compile with g++.*
- → ***./exe*** *# run program*

d. Test Example:



```cpp
#include <iostream>
#include "Timer.h"
#include <string>

using CLOCK = std::chrono::high_resolution_clock;
using TTimerCallback = std::function<void()>;

static CLOCK::time_point T0;

void logCallback(int id, const std::string &logstr) {
    auto dt = CLOCK::now() - T0;
    std::cout << "[" << std::chrono::duration_cast<std::chrono::milliseconds>(dt).count()
            << "] (cb " << id << "): " << logstr << std::endl;
}

int main(){
    Timer timer;
    std::this_thread::sleep_for(std::chrono::seconds(1));
    T0 = CLOCK::now();

    logCallback(-1, "main starting.");
    auto t1 = CLOCK::now() + std::chrono::seconds(1);
    auto t2 = t1 + std::chrono::seconds(1);

    timer.registerTimer(t2, [&]() {logCallback(1, "callback str"); });
    timer.registerTimer(t1, [&]() {logCallback(2, "callback str"); });
    timer.registerTimer(Millisecs(700), [&]() { logCallback(3, "callback str"); });
    timer.registerTimer(t1 + Millisecs(300), Millisecs(500), [&]() { logCallback(4, "callback str"); });
    timer.registerTimer([&]() {
        static int count = 0;
        return ++count < 3;
    }, Millisecs(500), [&]() { logCallback(5, "callback str"); });
    std::this_thread::sleep_for(std::chrono::seconds(5));
    logCallback(-1, "main terminating.");
}
```



```
furkan@furkan:~/Desktop/HW1$ ./exe
Timer::thread function starting...
[0] (cb -1): main starting.
[500] (cb 4): callback str
[500] (cb 5): callback str
[700] (cb 3): callback str
[1000] (cb 2): callback str
[1000] (cb 4): callback str
[1000] (cb 5): callback str
[1400] (cb 3): callback str
[1500] (cb 5): callback str
[2000] (cb 1): callback str
[2100] (cb 3): callback str
[2800] (cb 3): callback str
[3501] (cb 3): callback str
[4201] (cb 3): callback str
[4901] (cb 3): callback str
[5000] (cb -1): main terminating.
Timer::thread function terminating...
furkan@furkan:~/Desktop/HW1$
```