

# CSE 437 REAL TIME SYSTEM ARCHITECTURES

## HOMEWORK (Due 22<sup>th</sup> May 2023)

Design a timer event generator in C++.

1. Design a thread-safe C++ class that implements the following interface. The class will have its own thread to provide the timing functionality. All the callbacks will be called from this single timer thread.

```
using CLOCK = std::chrono::high_resolution_clock;
using TTimerCallback = std::function<void()>;
using Millisecs = std::chrono::milliseconds;
using Timepoint = CLOCK::time_point;
using TPredicate = std::function<bool()>;

class ITimer {
public:
    // run the callback once at time point tp.
    virtual void registerTimer(const Timepoint& tp, const TTimerCallback& cb) = 0;
    // run the callback periodically forever. The first call will be executed after the first period.
    virtual void registerTimer(const Millisecs& period, const TTimerCallback& cb) = 0;
    // Run the callback periodically until time point tp. The first call will be executed after the first period.
    virtual void registerTimer(const Timepoint& tp, const Millisecs& period, const TTimerCallback& cb) = 0;
    // Run the callback periodically. After calling the callback every time, call the predicate to check if the
    // termination criterion is satisfied. If the predicate returns false, stop calling the callback.
    virtual void registerTimer(const TPredicate& pred, const Millisecs& period, const TTimerCallback& cb) = 0;
};
```

2. Write a 2-page report containing descriptions of:
  - a. The requirements of your timer, containing constraints and assumptions.
  - b. The design of the timer
  - c. How to build and test your project
3. Use can use the example test application and its output given in the next page.

Notes:

- The designed C++ project is going to be built with a Makefile.
- Required threading, timing and synchronization features should be implemented with native C++ (11, 14, ...). No OS specific function calls are allowed (i.e. posix threads in linux, multimedia timer in Windows...)
- A “good” solution will get the timer thread to block until either the timer queue changes, or a timeout occurs, rather than blocking with a constant period (Think what happens if registerTimer is called while the thread is asleep).
- Your solution will be evaluated not only in terms of requirements, but also performance.
- The homework should be archived in a zip file (or a tar file) and uploaded to CSE 437 team. The archive should only contain the report, source files and Makefile. It shouldn't contain compiled binaries.

## Example Test Application

```
#include <iostream>
#include "Timer.h"
#include <string>
using CLOCK = std::chrono::high_resolution_clock;
using TTimerCallback = std::function<void()>;

static CLOCK::time_point T0;

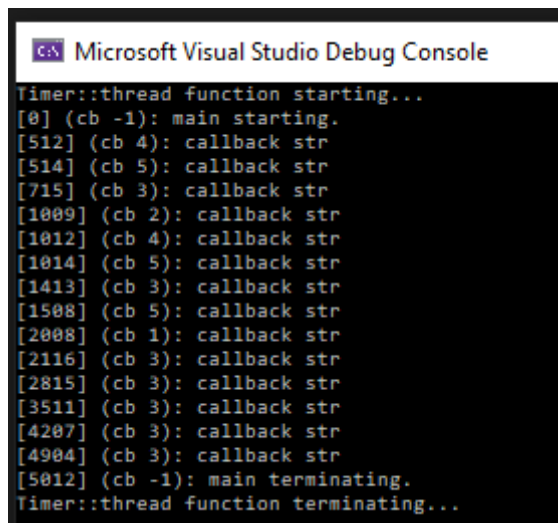
void logCallback(int id, const std::string &logstr) {
    auto dt = CLOCK::now() - T0;
    std::cout << "[" << std::chrono::duration_cast<std::chrono::milliseconds>(dt).count()
        << "] (cb " << id << "): " << logstr << std::endl;
}

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



```
Microsoft Visual Studio Debug Console
Timer::thread function starting...
[0] (cb -1): main starting.
[512] (cb 4): callback str
[514] (cb 5): callback str
[715] (cb 3): callback str
[1009] (cb 2): callback str
[1012] (cb 4): callback str
[1014] (cb 5): callback str
[1413] (cb 3): callback str
[1508] (cb 5): callback str
[2008] (cb 1): callback str
[2116] (cb 3): callback str
[2815] (cb 3): callback str
[3511] (cb 3): callback str
[4207] (cb 3): callback str
[4904] (cb 3): callback str
[5012] (cb -1): main terminating.
Timer::thread function terminating...
```