```cpp
#pragma once
#include<functional>
#include<memory>
#include<set>
#include<list>
#include<boost/asio.hpp>


namespace cpp_tools {

    class TimerEventGenerator;
    typedef std::shared_ptr<TimerEventGenerator> TimerEventGeneratorPtr;

    struct TimerEventGenerator : public
      std::enable_shared_from_this<TimerEventGenerator> {
        typedef std::function<void(void)> HandlerT;
        typedef std::shared_ptr<boost::asio::deadline_timer> TimerPtr;
        typedef std::set<TimerPtr> TimersT;
        typedef boost::posix_time::ptime         TimeT;
        typedef boost::posix_time::time_duration DurationT;

        static TimerEventGeneratorPtr make();
        void start();

        void add_timer(TimeT time, HandlerT handler);// execute handler one
          time at time if not stopped before. Must be called before start();
        void add_timer(std::string at_utc_time, HandlerT handler);// e.g.
          "2018-03-05 16:34:00.000"

        void add_timer(DurationT duration, HandlerT handler);// execute handler
           every duration until stopped. Must be called before start();
        void add_timer(size_t ms_duration, HandlerT handler);// duration in
          milliseconds
        void stop();

    private:
        TimerEventGenerator();
        void reset_timer(TimerPtr timer, DurationT duration, HandlerT handler);

    private:
        typedef
          boost::asio::executor_work_guard<boost::asio::io_context::executor_ty
          pe> io_context_work;
        std::shared_ptr<boost::asio::io_context >   m_context
          { std::make_shared<boost::asio::io_context>(1) };// run in 1 thread
        std::list<io_context_work>                  m_work;
        std::mutex                                  m_add_mutex;
        std::mutex                                  m_stop_mutex;

```

```
42          TimersT                                                 m_timers;
43      };
44
45
46  }
47
48
49  //////////////////////
50  /*
51  * time_event_generator.cpp
52  *
53  *  Created on: Mar 25, 2018
54  *      Author: ihersht
55  */
56
57  #include <boost/date_time/posix_time/posix_time.hpp>
58  #include "time_event_generator.h"
59
60
61  namespace cpp_tools {
62
63      TimerEventGeneratorPtr TimerEventGenerator::make() {
64          return TimerEventGeneratorPtr(new TimerEventGenerator);
65      }
66
67      TimerEventGenerator::TimerEventGenerator() {
68          m_work.push_back(boost::asio::make_work_guard(*m_context));
69      }
70
71      void TimerEventGenerator::start() {
72          boost::system::error_code ec;
73          m_context->run(ec);
74          if (ec) {
75              //log error
76          }
77      }
78
79      void TimerEventGenerator::stop() {
80          std::unique_lock<std::mutex> lc(m_stop_mutex);
81          if (m_context->stopped()) {
82              return;
83          }
84          boost::system::error_code ec;
85          for (auto & timer : m_timers) {
86              timer->cancel(ec);
87              if (ec) {
88                  //log error
89              }
90          }
```

```
 91          m_timers.clear();
 92          m_context->stop();
 93      }
 94
 95      void TimerEventGenerator::add_timer(TimeT time, HandlerT handler) {
 96          TimerPtr timer = std::make_shared<boost::asio::deadline_timer>
               (*m_context);
 97          timer->expires_at(time);
 98          {
 99              std::unique_lock<std::mutex> lc(m_add_mutex);
100              m_timers.insert(timer);
101          }
102          auto self(shared_from_this());
103          timer->async_wait([this, self, handler](const
               boost::system::error_code& ec) {
104              if (!ec) {
105                  try {
106                      handler();
107                  }
108                  catch (...) {
109                      // log error
110                      stop();
111                  }
112              }
113              else {
114                  // log error
115              }
116          }
117          );
118      }
119
120      void TimerEventGenerator::add_timer(std::string at_utc_time, HandlerT
           handler) {
121          TimeT time(boost::posix_time::time_from_string(at_utc_time));
122          add_timer(time, handler);
123      }
124
125
126      void TimerEventGenerator::add_timer(DurationT duration, HandlerT handler) {
127          TimerPtr timer = std::make_shared<boost::asio::deadline_timer>
               (*m_context);
128          {
129              std::unique_lock<std::mutex> lc(m_add_mutex);
130              m_timers.insert(timer);
131          }
132          reset_timer(timer, duration, handler);
133      }
134
135      void TimerEventGenerator::add_timer(size_t ms_duration, HandlerT handler) {
```

```
136         DurationT  duration = boost::posix_time::milliseconds(ms_duration);
137         add_timer(duration, handler);
138     }
139
140
141     void TimerEventGenerator::reset_timer(TimerPtr timer, DurationT duration,
          HandlerT handler) {
142         timer->expires_from_now(duration);
143         auto self(shared_from_this());
144         timer->async_wait([this, self, timer, duration, handler](const
            boost::system::error_code& ec) {
145             if (!ec) {
146                 try {
147                     handler();
148                     reset_timer(timer, duration, handler);
149                 }
150                 catch (...) {
151                     //log error
152                     stop();
153                 }
154
155             }
156             else {
157                 // log error
158             }
159         }
160         );
161
162     }
163
164
165 }
166 ////////////////////////////////////////////////////////
167 #include "time_event_generator.h"
168 #include<iostream>
169 #include<chrono>
170 static std::mutex io_mutex;
171 void th1() {
172     std::unique_lock<std::mutex> lc(io_mutex);
173     std::cout << "th1" << std::endl;
174 }
175 void th2() {
176     std::unique_lock<std::mutex> lc(io_mutex);
177     std::cout << "th2" << std::endl;
178 }
179
180 void dh1() {
181     std::unique_lock<std::mutex> lc(io_mutex);
182     std::cout << "dh1" << std::endl;
```

```
183 }
184 void dh2() {
185     std::unique_lock<std::mutex> lc(io_mutex);
186     std::cout << "dh2" << std::endl;
187 }
188
189
190 int main() {
191
192     cpp_tools::TimerEventGeneratorPtr timer =
          cpp_tools::TimerEventGenerator::make();
193
194     auto time = boost::posix_time::second_clock::universal_time();
195     timer->add_timer(time + boost::posix_time::seconds(5), th1);
196     timer->add_timer(time + boost::posix_time::seconds(7), th2);
197     timer->add_timer(boost::posix_time::seconds(1), dh1);
198     timer->add_timer(boost::posix_time::seconds(2), dh2);
199
200     std::thread th([timer] {timer->start(); });
201     std::this_thread::sleep_for(std::chrono::seconds(31));
202     timer->stop();
203     th.join();
204
205     return 0;
206 }
207
```