CSE 437 Realtime Systems HW1
Timer Event Generator Report
Bahadir Etka Kilinc
1901042701

a. Requirements of the Timer

The requirements for this timer application are drawn from a need for a robust, versatile and highly customizable timer system that can work independently from the main application thread. A thorough understanding of these requirements and constraints provides the foundation for this project.
Requirements:
Multiple Timing Events: The timer is required to handle multiple timing events concurrently. This is a crucial feature that allows the timer to be versatile and useful in complex applications where multiple independent events need to be tracked.
Periodic Timing Events: The timer must be capable of handling periodic timing events. This feature allows events to be executed periodically, repeating after a specified time interval. This could be used in scenarios like regular data updates, health checks in software systems, and so on.
One-Time Timing Events: The timer must be capable of handling one-time timing events. These are events that happen once at a specified point in time in the future. This is useful for scenarios like timeouts or scheduling a task for a future time.
Conditional Timing Events: The timer should support timing events that happen based on a condition. These events keep happening periodically as long as a condition holds true.
Thread-Independent Operation: The timer is required to run independently on its own thread. This ensures that the main application thread is not blocked or slowed down by the timer, providing an asynchronous approach to handling events.
Callback Execution: The timer should allow users to specify a callback function, i.e., an action to be executed when the timing event occurs. This provides flexibility in handling the timer event when it is triggered.
Constraints:
-The timer's functionality is heavily dependent on the system clock. Hence, the precision, accuracy, and performance of the system clock can directly impact the timer's effectiveness.
Assumptions:
-The callback functions provided by the users do not throw exceptions. Handling exceptions within the timer thread is considered beyond the scope of this timer implementation.

b. Design of the Timer

The timer application is designed with the emphasis on object-oriented principles and ease of use for the end-user. It consists of several interconnected components that work together to provide a versatile and robust timer functionality.
ITimer Interface: This interface defines the basic timer operations, which include four types of timer registration methods. The ITimer interface serves as a contract for any timer class implementation, thereby providing a way to ensure that certain functionalities are present in the timer class.

Timer Class: The Timer class is the heart of the timer application. It implements the ITimer interface and provides the primary functionality of the timer. The Timer class is designed to manage and execute multiple timing events concurrently, all within its dedicated thread.
TimerEvent Struct: This structure defines the necessary attributes for a timer event: next execution time, period, callback function, and a predicate function for conditional events. This structure is used by the Timer class to manage each registered timer.
TimerEventComparison Class: This helper class is used to manage the priority queue that holds TimerEvent objects. It ensures that the timer events are organized based on their next execution time. The event due to happen the soonest is always at the top of the queue.

c. Instructions on Building and Testing the Project

Building the Project:
The project is constructed to be easily compiled and built using the GNU **make** utility. A **Makefile** is included in the project directory, which defines the rules to compile the source code and link the resulting object files to create the final executable.
To build the project, navigate to the project directory in the terminal and execute the **make** command. This command uses the **Makefile** to automate the build process, and creates an executable named **timer**.
Testing the Project:
The timer application is designed to be easily testable. The main testing file is **main.cpp**, which includes a variety of test cases for the Timer class. These test cases demonstrate the core functionality and versatility of the Timer class.
To test the project, execute the **timer** binary created during the build process. The output displays the times at which different timing events occurred and the actions executed when the events were triggered.
New test cases can be added in **main.cpp** by calling the registerTimer methods with different parameters, and creating new callback functions to be executed when the timing events occur. It's crucial to provide detailed comments with each test case, explaining its purpose and expected outcome. This practice increases the readability and maintainability of the test suite, making it easier for other developers to understand and extend the testing environment.
This robust timer application, with its well-defined requirements, thoughtful design, and comprehensive testing strategy, provides a versatile tool that can be integrated into a wide range of software systems.