ASSIGNMENT – 2
POSIX-THREADS - ALARM APP
(Lab – Report)

Course Number and Name: EECS 3221 - OPERATING SYSTEM	
Name of Students: 1) Sharma, Akalpit : akalpit@my.yorku.ca 2) Lo, Jacky : jackygsl@my.yorku.ca 3) Panjawani, Novej : novej@my.yorku.ca 4) Li, Chaohan : chaohan@my.yorku.ca 5) Lin, Yun : lyjohn@my.yorku.ca 6) Singh, Mandeep sngm1190@my.yorku.ca	**Course Director:** Mr. Jia Xu
Date of Report Submitted: February 25, 2019	**Grade:**

Objectives:

To design and develop a multi-threaded alarm displaying application.

Problem Analysis:

Following were the gist of the problem specification-

- A multi-threaded alarm displaying system
- 2 types of alarm related command -
  - start – to start an alarm in a thread
  - end – to stop/end an already scheduled alarm
- Proper messages to inform the user whats happening right including-
  - Create, Insert and Start a new alarm thread
  - End an already scheduled alarm thread
- Detection capability for invalid command

Solution:

To solve the problem the following design decisions were made -

1. Pre-allocated alarm slots
2. Alarm threads will be created as needed
3. Alarm threads will have some means to inform the main thread of their state (finishing/ending)
4. A comprehensive help menu to aknowledge the user with the available commands
5. A quit command to exit the program at any time
6. Usage of mutex to synchronize the alarm list
7. Free-up allocated memories for alarm datastructures

To simplify the solution, the number of alarm slots have been pre-defined which can be changed by setting the macro NUM_ALARMS and re-building the program.

So when user starts the program, it will wait for a new start command and get alarm parameters from the user input. Then if there is an empty alarm slot, it will configure the alarm data structure, start a new thread to execute it and get back to the prompt for new commands. If user inserts an end command it will get the id of the alarm to end and upong successful finding, the main thread will schedule the assigned thread to stop or end. User can enter quit command at any time to exit the program. If user enters an end command the program will first schedule all running threads to stop and wait for them to finish. Afterwards the main thread will free up the allocated memory spaces and exit saying a good bye message.

Algorithm:

1. start the program
2. main thread allocates alarm list
3. main thread goes into loop
4. main thread asks user for new command

5. if user enters start command-
   5.1.main thread reads the alarm duration and message
   5.2.main thread checks if any empty slot is available
   5.3.if empty slot is available -
      a) main thread creates a new thread and assigns it to the new alarm
      b) child thread starts working
         - child thead goes into a loop
         - it checks the alarm list for the assigned id
         - if id is not found it exits
         - if id is found it checks for the duration and elapsed time
         - if the current time is greater than or equal to the alarm time it displays the alarm and exits
         - if main thread has requested/scheduled to end the alarm it exits
         - if current time is less than the alarm time it sleeps for 1 second and continues the loop
   5.4.if theres no empty slot main thread politely asks the user to wait for an alarm to finish first

6. if user enters end command-
   6.1.main thread gets the alarm id from user
   6.2.if alarm id is not found in the list informs the user
   6.3.if alarm id is found in the list-
      a) main thread schedules the alarm to be ended in the next iteration of the designated child thread
      b) eventually the child thread gets the flag and exits

7. if user enters help command main thread shows the help menu

8. if user enters quit command main thread breaks the loop and -
   8.1.schedules any running alarm displaying thread to end/finish
   8.2.waits for the threads to finish
   8.3.frees up the allocated memory

9. finally main exits