EECS3221M Assignment 3

Instructor: Jia Xu
Date: April 6, 2020

Edward Izraitel, 214964225
Jae Don Choi, 214640668
Solang Mun, 216826331
Seohyun Jeong, 215322092

Introduction

This assignment was an opportunity to learn about the use of POSIX threads for a basic multithreading program and how to coordinate the threads in order to avoid issues such as the Readers-Writers Problem. POSIX threads were synchronized using semaphores and condition variables. This was implemented in New_Alarm_Cond.c. Specifically our focus was on the first and the second readers-writers problems: that no reader be kept waiting unless a writer has already obtained permission to use the shared object and that once a writer is ready, that writer performs its write as soon as possible, respectively

Design and Implementation of Requirements

A3.1

The requirements of A3.1 are for two types of alarm requests to be recognized and handled. Some modifications were made to the main method to meet these requirements. Alarm ID and Group ID fields were added to the alarm struct to store the required input data. The while loop in the main method was expanded to include the new requests "Start_Alarm" and "Change_Alarm".

A3.2

In A3.2, the main thread checks if the each alarm request is consistent with the required format. If inconsistent, the request will be rejected prompting a message "Bad command". The main thread is a "writer" process as it adds alarms to the alarm list. It was necessary to utilize semaphores to ensure that the main thread was not interfering with "reader" processes and to ensure mutual exclusion between the main thread and the removal thread.

For A3.2.1, the main thread invokes the alarm_insert method with the new alarm as an argument to insert an alarm with the specified Alarm_ID into the alarm list sorted in the order of their Alarm_IDs for each Start_Alarm request received.

For A3.2.2, the main thread invokes the change_alarm method with the new alarm as an argument to replace the Group_ID, Time, and message for each Change_Alarm request. The change_alarm method goes through the list of alarms until it finds a matching Alarm_ID to assign the new Group_ID, time, and the message.

A3.3

The requirement specifies that the **alarm group display creation thread** is responsible for creating **display alarm threads** such that each display alarm thread will only print messages only for the alarms that have the same particular Group_ID.

The alarm group display creation thread is triggered by any new Start_Alarm request and Change_Alarm request where the Group_ID has changed. The **alarm group display**

creation thread is a “reader” process and readily accesses the alarm list given there is no writer that currently has obtained permission to use the alarm list.

Given an alarm, the **alarm group display creation thread** goes through the list that holds tuples of **display alarm thread** and Group_ID and searches for a **display alarm thread** with a matching Group_ID. If it fails to find such a thread, a new **display alarm thread** is created and a reference to the new thread forms a new tuple with the new Group_ID in the list.

However, if a **display alarm thread** with a matching Group_ID is found, the given alarm is assigned to that display alarm thread which will print the alarm message every 5 seconds until the alarm expires.

A3.4

The **alarm removal thread,** which is both a “reader” and a “writer”, loops through the alarm list to find alarms that are nearing expiry using the smallest() method as a “reader” process. Once it finds an alarm that is about to expire, it triggers the Display Alarm Thread to initiate A3.5.2 (removal message), then it must wait for the readers to finish reading before it gains control of the read-write semaphore and proceeds to “write” or remove an alarm and subsequently a removal method is displayed.

A3.5

A3.5.1 specifies that after each **display alarm thread** has been created by the **alarm group display creation thread,** the display alarm will print “Alarm (<alarm_id>) Printed by Alarm Display Thread <thread-id> at <print_time>: Group(<group_id>) <time message> every 5 seconds for each alarm with the same Group_ID. The **display alarm thread** is strictly a “reader” with regard to the shared alarm list and the reads are coordinated with “writers” using semaphores and a reader flag variable to indicate that a display alarm thread is reading the shared alarm list. The display alarm thread traverses through the alarm list and whenever it finds a matching Group_ID, it prints the default display message.

A3.5.2 requirements were met by accessing the alarm list in search of the Alarm_ID. The removal thread, just before removal, triggers the display alarm thread to display the removal message and provides the Alarm_ID. The Display Alarm Thread then goes through the alarm list to find the alarm with the matching Alarm_ID and prints the removal message. Then, the alarm is removed near simultaneously by the Alarm Removal Thread.

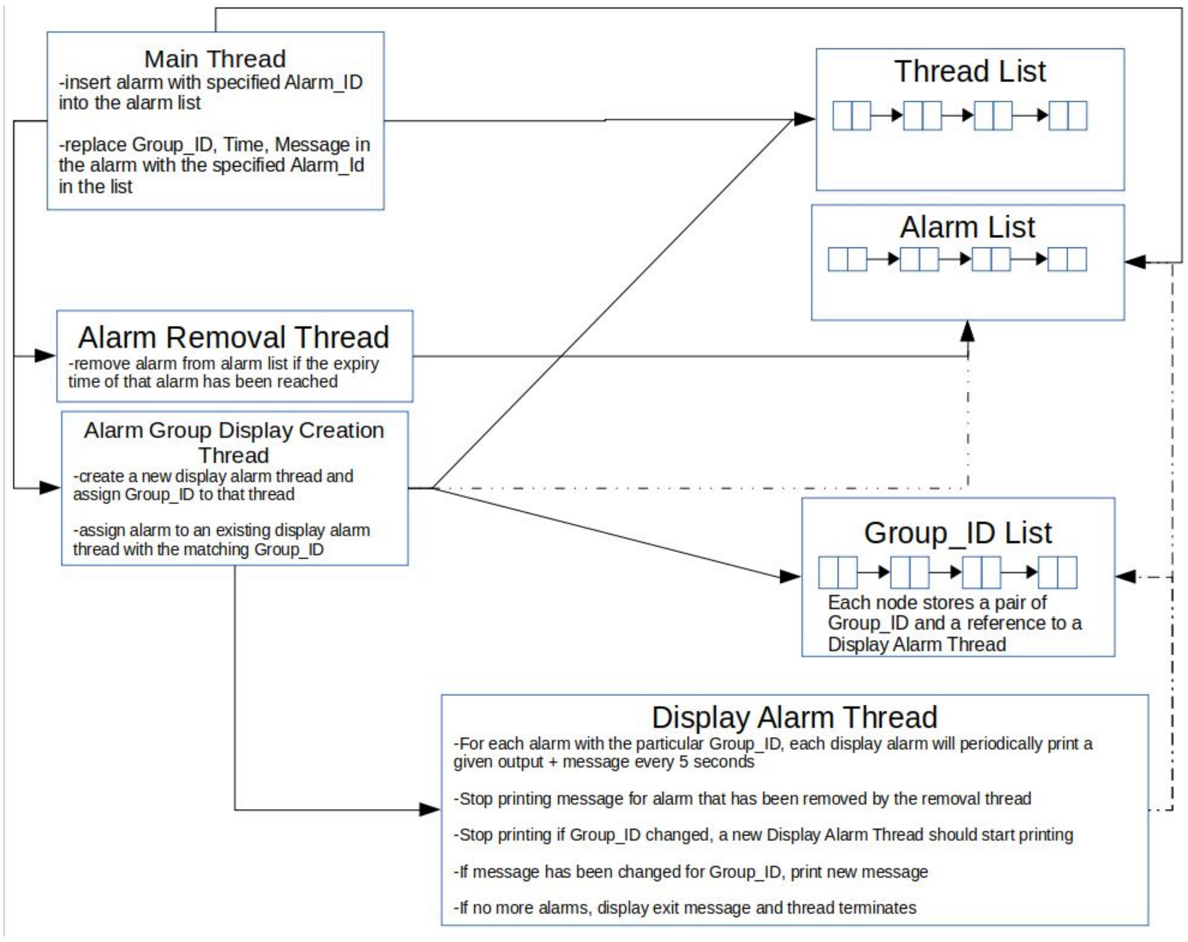
To satisfy A3.5.3 and A3.5.4 requirements for changes made to alarms, we have added a flag variable to indicate that a change has been made. Just before the main thread applies the changes to the alarms according to the Change_Alarm requests, it prompts the Display Alarm Thread to print the A3.5.3 message. Then, the main thread turns the change flag within the alarm to changed. Once the main thread has finished updating the alarm, the **alarm group display** determines if the Group_ID has been changed. If the Group_ID has been changed, then it either reassigns the alarm to the corresponding Display Alarm Thread

or creates a new Display Alarm Thread if there is no Display Alarm Thread for the new Group_ID.
Then, the display alarm thread finds that a change has been made to the alarm, the updated message will be displayed every 5 seconds.

A3.5.5 is responsible for the empty Display Alarm Threads. As such, every 5 seconds, when the thread checks for alarms to display messages for, it traverses through the alarm list looking for an alarm with a matching Group_ID. If there is no alarm in the alarm list with the given Group_ID, it prints the termination message as required and the thread will terminate shortly after.

A3.6.1 and A3.6.2

The requirement for synchronization being treated as solving a “Readers-Writers” problem is outlined in the diagram below where solid arrows indicate “Writer” processes and the dashed lines indicate “Reader” processes. The access to the shared alarm list were synchronized using unnamed semaphores.



Main Thread
-insert alarm with specified Alarm_ID into the alarm list
-replace Group_ID, Time, Message in the alarm with the specified Alarm_Id in the list
Thread List
Alarm List
Alarm Removal Thread
-remove alarm from alarm list if the expiry time of that alarm has been reached
Alarm Group Display Creation Thread
-create a new display alarm thread and assign Group_ID to that thread
-assign alarm to an existing display alarm thread with the matching Group_ID
Group_ID List
Each node stores a pair of Group_ID and a reference to a Display Alarm Thread
Display Alarm Thread
-For each alarm with the particular Group_ID, each display alarm will periodically print a given output + message every 5 seconds
-Stop printing message for alarm that has been removed by the removal thread
-Stop printing if Group_ID changed, a new Display Alarm Thread should start printing
-If message has been changed for Group_ID, print new message
-If no more alarms, display exit message and thread terminates

Source Code

See New_Alarm.Cond.c

Testing

A3.1a

1. “Start_Alarm” request functions properly given the command in the format “Start_Alarm(Alarm_ID): Group(Group_ID) Time Message”
2. “Start_Alarm” request without Alarm_ID generates an error message
3. “Start_Alarm” request without Group_ID generates an error message
4. “Start_Alarm” request without Time generates an error message

A3.1b

1. Given an existing alarm with a matching Alarm_ID, “Change_Alarm” request functions properly given the command Change_Alarm(Alarm_ID): Group(Group_ID) TIme Message with new: Group_ID, Time, and Message
2. Given an existing alarm with a matching Alarm_ID, “Change_Alarm” request functions properly given the command Change_Alarm(Alarm_ID): Group(Group_ID) TIme Message with the same Group_ID but different Time and Message
3. Given an existing alarm with a matching Alarm_ID, “Change_Alarm” request functions properly given the command Change_Alarm(Alarm_ID): Group(Group_ID) TIme Message with the same Group_ID and Time but a different Message
4. “Change_Alarm” request generates an error message when given a non-existing Alarm_ID.
5. “Change_Alarm” request generates an error message if any of the arguments is missing.

A3.2.1

1. Start_Alarm request generates the message “Alarm(<alarm_id>) Inserted by Main Thread <thread-id> Into ALarm List at <insert_time>: Group(<group_id>) <time message>”

A3.2.2

1. Given an existing alarm with a matching Alarm_ID and changed Group_ID and message, Change_Alarm request generates the message: “Alarm(<alarm_id>) Changed at <alarm_change_time>: Group(<group_id>) <time message>” with the changes.

A3.3a

1. Upon entry of a Start_Alarm request with a Group_ID that has not previously been entered, the alarm group creation thread creates a new display alarm thread and prints “Alarm Group Display Creation Thread Created New Display Alarm Thread <threadid> For Alarm(<alarm_id> at <create_time>: Group(<group_id>) <time message>”.
2. Upon entry of a Change_Alarm request with a Group_ID that has not previously been entered, a new display alarm thread is created but should print “Display Thread <thread-id> Has Taken Over Printing Message of Alarm(<alarm_id>at <new_group_change_time>: Changed Group(<group_id>) <time message>” instead.

A3.3b

1. Upon entry of a Start_Alarm request with a Group_ID that has previously been entered, the alarm group creation thread assigns the alarm to an existing display alarm thread and prints “Alarm Thread Display Alarm Thread <thread-id> Assigned to Display Alarm(<alarm_id> at <assign_time>: Group(<group_id>) < time message>”
2. Upon entry of a Change_Alarm request with a Group_ID that has previously been entered, the alarm is reassigned to another existing display alarm thread and should print “Display Thread <thread-id> Has Taken Over Printing Message of Alarm(<alarm_id>at <new_group_change_time>: Changed Group(<group_id>) <time message>” .

A3.4

1. A Start_Alarm request, “Start_Alarm(Alarm_ID): Group(Group_ID) Time Message”, is made which prints the message “Alarm Removal Thread Has Removed Alarm(<alarm_id> at <remove_time>: Group(<group_id>) <time message>” after the entered time.

A3.5.1

1. A Start_Alarm request causes the **alarm group display creation thread** to create a new display alarm thread that prints “Alarm (<alarm_id>) Printed by Alarm Display Thread <thread-id>at <print_time>:Group(<group_id>) <time message>” every 5 seconds.
2. A second Start_Alarm request with a new Group_ID causes the **alarm group display creation thread** to create a new display alarm thread that prints “Alarm (<alarm_id>) Printed by Alarm Display Thread <thread-id>at <print_time>:Group(<group_id>) <time message>” every 5 seconds resulting in an alternating print of the display message.

A3.5.2

1. A Start_Alarm request for an alarm set at 10 seconds expires and prints two separate messages: first message by the Alarm Removal Thread “Alarm Removal Thread Has Removed Alarm(<alarm_id> at <remove_time>:Group(<group_id>) <time

message>” followed by a second message by the display alarm thread to which the alarm belongs “Display Thread <thread-id> Has Stopped Printing Message of Alarm(<alarm_id>at<stopped_print_time>: Group(<group_id>) <time message>.” The thread should no longer print anything for the alarm when observed for over 5 seconds.

A3.5.3

1. A Start_Alarm request is made followed by a Change_Alarm request for a new Group_ID. The previous and the new display alarm threads display the following messages respectively: “Display Thread <thread-id> Has Stopped Printing Message of Alarm(<alarm_id>at<old_group_change_time>: Changed Group(<group_id>) <time message>” and “Display Thread <thread-id> Has Taken Over Printing Message of Alarm(<alarm_id>at <new_group_change_time>: Changed Group(<group_id>) <timemessage>”. The first message should not print again while the second message should repeat every 5 seconds until the alarm is expired.

A3.5.4

1. A Start_Alarm request is made followed by a Change_Alarm request for the same Group_ID. The Display Alarm Thread should print “Display Thread <thread-id> Starts to Print Changed Message Alarm(<alarm_id> at <message_change_time>: Group(<group_id>) < time message>”.

A3.5.5

1. Two alarms with the same Group_ID are started using the Start_Alarm request. Then the first alarm expires and results in the following two messages: Alarm Removal Thread Has Removed Alarm(<alarm_id> at <remove_time>:Group(<group_id>) <time message>” by the alarm **removal thread** and “Display Thread <thread-id> Has Stopped Printing Message of Alarm(<alarm_id>at <stopped_print_time>: Group(<group_id>) <time message>” by the **display alarm thread**. Then, when the second alarm expires, the above two messages should repeat, followed by a third message: No More Alarms in Group(<group_id>): Display Thread <thread-id> exiting at <exit_time>”.. The display alarm thread should no longer print anything.

Test Output

Testing

A3.1a → same as A3.2.1

1. Start_Alarm properly working.

```
Alarm> Start_Alarm(10): Group(50) 50 test1

Alarm(10) Inserted by Main Thread -1190711672 Into Alarm List at 1586143545: Group(50) test1
```

2. Start_Alarm request with wrong format.

```
Alarm> Start_Alarm: Group(10) 500 Wrongformat1 //no Alarm_Id

Bad Command

Alarm> Start_Alarm: Group() 500 Wrongformat2 //no Group_Id

Bad Command

Alarm> Start_Alarm(10): Group(20) 50 //no message

Bad Command

Alarm> Start_Alarm(10): Group(20) Wrongformat4 //no time

Bad Command
```

A3.1b → same as A3.2.2

1. Change_Alarm properly working.

```
Alarm> Start_Alarm(10): Group(10) 50 Test1
```

```
Alarm(10) Inserted by Main Thread 399906376 Into Alarm List at 1586144078: Group(10)
Test1

Alarm> Change_Alarm(10): Group(20) 50 Change1

Display Thread <thread-id> Has Stopped Printing Message of Alarm(10) at 1586144128:
Changed Group(20) Change1

Alarm(10) Changed at 1586144096: Group(20) Change1
```

2. Change_Alarm wrong format

```
Alarm> Change_Alarm: Group(10) 10 Wrongformat1 //no Alarm_Id

Bad Command

Alarm> Change_Alarm(10): Group 50 Wrongformat2 //no Group_Id

Bad Command

Alarm> Change_Alarm(10): Group(50) 50 //no message

Bad Command

Alarm> Change_Alarm(10): Group(50) Wrongformat4 //no time

Bad Command
```

A3.3A (NOT Possible we do not have display thread responsible for this action)

A3.3A (NOT Possible we do not have remove methods removing alarms that are expired)

A3.5

1. Same Alarm_ID, BUT different Group_ID for Change request.

```
Alarm> Start_Alarm(50): Group(50) 500 Test
```

```
Alarm(50) Inserted by Main Thread 1883273000 Into Alarm List at 1586144841: Group(50) Test

Alarm> Change_Alarm(50): Group(10) 500 differentGroup

Display Thread <thread-id> Has Stopped Printing Message of Alarm(50) at 1586145341: Changed Group(10) differentGroup
```

2. Same Alarm_ID, AND same Group_ID

```
Alarm> Start_Alarm(10): Group(10) 500 Test2

Alarm(10) Inserted by Main Thread 1883273000 Into Alarm List at 1586144904: Group(10) Test2

Alarm> Change_Alarm(10): Group(10) 500 sameGroup

Alarm(10) Changed at 1586144927: Group(10) sameGroup
```

Readme

1. First copy the files "New_Alarm_Cond.c", and "errors.h" into your own directory.

2. To compile the program "alarm_cond.c", use the following command:

```
cc New_Alarm_Cond.c -D_POSIX_PTHREAD_SEMANTICS -lpthread
```

3. Type "a.out" to run the executable code.

4. At the prompt "ALARM>", two commands are available:
   - Start_Alarm with the syntax Alarm> Start_Alarm(Alarm_ID): Group(Group_ID) Time Message, where
     - Alarm_ID, Group_ID, and Time are positive integer inputs, and
     - Message is any string of length up to 128 characters
   - Change_Alarm with the syntax Alarm> Change_Alarm(Alarm_ID): Group(Group_ID) Time Message

Ex.

```
  ALARM> Start_Alarm(2345): Group(13) 50 Will meet you at Grandma’s house at 6pm.
```

Or

```
  ALARM> Change_Alarm(2345): Group(21) 80 Will meet you at Grandma’s house later at 8 pm
```

If the user types in something other than one of the above two types of valid alarm requests, then an error message will be displayed, and the invalid request will be discarded.

(To exit from the program, type Ctrl-d.)