EECS3221 Winter 2020
Assignment 2

POSIX Threads

Due Date: Monday, February 24, 2020, 8:30 a.m.

A. Description of the Assignment

A1. You are required to read and fully understand the first 4 chapters, that is, pages 1-129, of the book "Programming with POSIX Threads" by (This book is currently on reserve at Steacie Science Library, Call Number QA 76.76 T55 B88 1997).

A2. You are required to download the program "alarm_mutex.c" and the file "errors.h" and "README" from the directory /cs/course/3221M, and then try to compile and execute this program by following the instructions in the "README" file. (This program is explained in pages 52-58 of the book by David R. Butenhof).

A3. You are required to make the following changes to the program "alarm_mutex.c" to produce a program named "New_Alarm_Mutex.c".

A3.1 Two types of "Alarm requests" in "New_Alarm_Mutex.c"

Two types of "Alarm requests", "Start_Alarm", and "Change_Alarm" requests, are recognized and handled by "New_Alarm_Mutex.c", as follows:

(a) "Start_Alarm" requests with the following format:

Alarm> Start_Alarm(Alarm_ID): Time Message

- "Start_Alarm" is a keyword.
- "Alarm_ID" is a positive integer.
- "Time" has the same meaning and syntax as in the program "alarm_mutex.c".
- "Message" has the same meaning and syntax as in the program "alarm_mutex.c".

For example:
Alarm> Start_Alarm(2345): 50 Will meet you at Grandma's house at 6 pm

(b) "Change_Alarm" requests with the following format:

Alarm> Change_Alarm(Alarm_ID): Time Message

- "Change_Alarm" is a keyword.
- "Alarm_ID" is a positive integer.
- "Time" has the same meaning and syntax as in the program "alarm_mutex.c".
- "Message" has the same meaning and syntax as in the program "alarm_mutex.c".

For example:
Alarm> Change_Alarm(2345): 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.

A3.2. The main thread in "New_Alarm_Mutex.c"

The main thread in "New_Alarm_Mutex.c" will first determine whether the format of each alarm request is consistent with the formats specified above; otherwise the alarm request will be rejected with an error message. If a Message exceeds 128 characters, it will be truncated to 128 characters.

A.3.2.1. For each Start_Alarm request received, the main thread will **insert** the corresponding alarm with the specified Alarm_ID into the alarm list, *in which all the alarms are placed in the order of their Alarm_IDs.* Then the main thread will print:
"Alarm(<alarm_id>) Inserted by Main Thread <thread-id> Into Alarm List at <insert_time>: <time message>".

Note that above, and in each of the messages printed by the various threads below, <thread-id> is the thread identifier>; <insert_time>, <create_time>, <assign_time>, <remove_time>, <change_time>, <display_change_time>, <terminate_time>, <print_time>, etc., are the actual times at which the corresponding action was performed by each thread; those times are all expressed as the number of seconds from the Unix Epoch Jan 1 1970 00:00; <time message> correspond to "Time" and "Message" as specified in A3.1 (a), (b) above.

A.3.2.2. For each Change_Alarm request received, the main thread will use the specified Time and Message values in the Change_Alarm request to replace the Time and Message values in the alarm with the specified Alarm_Id in the alarm list. Then the main thread will print:
Alarm(<alarm_id>) Changed at <change_time>: <time message>".

A3.3. The alarm_thread in "New_Alarm_Mutex.c"

A.3.3.1. For each newly inserted alarm in the alarm list, the alarm thread will do the following:

If the number of existing display alarm threads is less than 3,
Then: (a) create a new display alarm thread and assign that alarm to the newly created display alarm thread. Then the alarm thread will print:
"Alarm Thread Created New Display Alarm Thread <thread-id> For Alarm(<alarm_id> at <create_time>: <time message>".

Else: (b) assign that alarm to an existing display alarm thread which has the smallest number of alarms assigned to it among the 3 existing display alarm threads. Then the alarm thread will print:
"Alarm Thread Display Alarm Thread <thread-id> Assigned to Display Alarm(<alarm_id> at <assign_time>: <time message>".

A.3.3.2. For each alarm in the alarm list, **if the expiry time of that alarm has been reached**, then the alarm thread will **remove that alarm from the alarm list, and the associated display alarm thread will stop printing the message in that alarm** Then the alarm thread will print:
"Alarm Thread Removed Alarm(<alarm_id> at <remove_time>: <time message>".

A.3.3.3. For each alarm in the alarm list, **if the expiry time of that alarm has been reached and the corresponding display alarm thread does not have any more alarms assigned to it, then terminate that display alarm thread**. Then the alarm thread will print:
"Alarm Thread Terminated Display Thread <thread-id> at <terminate_time>".

A3.4. The display alarm_threads in "New_Alarm_Mutex.c"

A3.4.1. After each display alarm thread in "New_Alarm_Mutex.c" has been created by the alarm thread, that display alarm thread will periodically print, every 5 seconds, the following:

"***Alarm (<alarm_id>) Printed by Alarm Display Thread <thread-id>at <print_time>: <time message>*** ".

A.3.4.2. For each newly changed alarm in the alarm list, **the display alarm thread associated with that alarm will start** periodically printing, every 5 seconds, the new message in that newly changed alarm. When the display alarm thread first starts printing a new changed message it will print:
"Display Thread <thread-id> Starts to Print Changed Message at <display_change_time>: <time message>".

B. Platform on Which The Programs Are to be Implemented

The programs should to be implemented using the ANSI C programming language and using the Linux system at York. You should use POSIX system calls or POSIX functions whenever possible.

C. Additional Requirements

(a) You must make sure that your code has very detailed comments.

(b) You must make sure that your code compiles correctly with your Make file.

(c) You must make sure that your code does not generate segmentation faults.

(d) You must make sure that your code is able to handle incorrect input.

(e) You must describe in detail any problems or difficulties that you had encountered, and how you solved or were able to overcome those problems or difficulties in the report.

Failure to satisfy the additional requirements above will result in a very low mark for the assignment.

D. <u>What to Hand In</u>

Each group is required to hand in both a hard copy and an electronic copy of the following:

1. A written report that identifies and addresses all the important aspects and issues in the design and implementation of the programs for the problem described above.

2. The C source programs.

3. A “Test_output” file containing the output of any testing your group has done.

4. A “makefile” file to make it easier for the marker to compile and run your group’s program.

5. A “README” file explaining how to compile and run your group’s program.

Each group is required to use the utility "submit" to submit the electronic version of the above 5 files plus the “errors.h” file to the course directory /cs/course/3221M/submit/a2

(The file “errors.h” should be included among the files submitted so that the marker can test whether your group’s programs can compile and run correctly or not.)

Important Warning:

Only submitting an electronic copy of your group’s assignment is not enough! If your group fails to submit a hard copy of your assignment on or before the due date, your group’s assignment will receive a grade of ‘F’.

E. <u>Evaluation of the Assignment</u>

1. The report part of your group's assignment (50%) will be evaluated according to:

(a) Whether all important design and implementation aspects and issues of your group's programs related to the problem above have been identified and appropriately addressed.

(b) How well you have justified your design decisions.

(c) The quality of your design.

(d) How well you have designed and explained the testing.

(e) The clarity, and readability of the report.

2. The program and testing part of your assignment (50%) will be evaluated according to:

(a) The quality of the design and implementation of your programs.

(b) The quality of the testing of your programs.

(c) Whether your programs satisfy the Additional Requirements in section C above.

F. Notes

Please note that the requirements specified in section A. Description of the Assignment above, are the *minimum requirements* that must be satisfied by your program. Obviously, there are many other possible details of the alarm system that have been left unspecified. It is your responsibility to make appropriate design and implementation choices concerning the unspecified details of the alarm system, and justify those decisions in your report.