EECS3221 Section E Fall 2024
Assignment 2

POSIX Threads

Due Date: Monday, November 4, 2024, 23:59 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/3221E/assign2 on the Red server at York (these files are also available on eClass), 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).

((Please direct all inquiries about how to login to the Red server to the Helpdesk at York University Information Technologies (UIT). You may also ask any EECS Lab monitor on the first floor of Lassonde Building about how to login to the Red server. Once you have logged into the Red server, in order to learn how to use any command such as “submit” on the Red server, type “submit”.)

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

There are 3 different types of threads in “new_alarm_mutex.c”:

(a) A main thread;
(b) An alarm thread;
(c) Display threads.

An “alarm list” is the only data structure that is shared between the 3 different types of threads: main thread, alarm thread, and display threads.

A3.1 Four types of “Alarm requests” in “new_alarm_mutex.c”

Four types of “Alarm requests”, “Start_Alarm”, “Change_Alarm”, “Cancel_Alarm” and “View_Alarms“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): Type Time Message

- “Start_Alarm” is a keyword.
- “Alarm_ID” is a positive integer.
- “Type” is a character string that starts with the character “T’.
- “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): T1 50 Will meet you at Grandma’s house at 6 pm

(b) “Change_Alarm” requests with the following format:

Alarm> Change_Alarm(Alarm_ID): Type Time Message

- “Change_Alarm” is a keyword.
- “Alarm_ID” is a positive integer.
- “Type” is a character string that starts with the character “T’.
- “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): T2 80 Will meet you at Grandma’s house later at 8 pm

(c) “Cancel_Alarm” requests with the following format:

Alarm> Cancel_Alarm(Alarm_ID): Type Time Message

- “Change_Alarm” is a keyword.
- “Alarm_ID” is a positive integer.

For example:
Alarm> Cancel_Alarm(2345)

(d) “View_Alarms” requests with the following format:

Alarm> View_Alarms

- “View_Alarms” is a keyword.

For example:
Alarm> View_Alarms

If the user types in something other than one of the above four 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" first creates an alarm thread. The alarm thread is described in section A.3.3 further below.

For each alarm request, the main thread in "New_alarm_mutex.c" will first check whether the format of the 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 valid **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>, <change_time>, <cancel_time>, <creation_time>, <assign_time>,<display_change_time>, <stop_display_time>, <terminate_time>,<periodic_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 UnixEpoch Jan 1 1970 00:00; <type time message> correspond to "Type", "Time" and "Message" as specified in A3.1 (a), (b) above.

A.3.2.2. For each valid **Change_Alarm** request received, the main thread will use the specified Type, Time and Message values in the Change_Alarm request to replace the Type, 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>: <type time message>*"**.

A.3.2.3. For each valid **Cancel_Alarm** request received, the main thread will remove the alarm with the specified Alarm_Id from the alarm list. Then the main thread will print: ***Alarm(<alarm_id>) Cancelled at <cancel_time>: <type time message>*"**.

A3.2.4. For each alarm in the alarm list, if the specified number of n seconds has expired, then the main thread will remove that alarm from the alarm list, and it will print: "***Alarm(<alarm_id>): Alarm Expired at <time>: Alarm Removed From Alarm List*** ", where <time> is the actual time at which this was printed (<time> is expressed as the number of seconds from the Unix Epoch Jan 1 1970 00:00.

A3.2.5. For each **View_Alarms** request received, the main thread will print out the following:

- A list of all the current existing display threads, together with the alarms in the alarm list that the alarm thread has assigned to each display thread, in the following format:
"View Alarms at <view_time>: <time>:
1. Display Thread <thread-id> Assigned:
1a. Alarm(<alarm_id>): <type time message>
1b. Alarm(<alarm_id>): <type time message> **(if a second alarm is assigned)**
2. Display Thread <thread-id> Assigned:
2a. Alarm(<alarm_id>): <type time message>
2b. Alarm(<alarm_id>): <type time message> **(if a second alarm is assigned)**
…"

A3.3. <u>The alarm thread in "new_alarm_mutex.c"</u>

After the alarm thread has been created by the main thread, it will do the following:

A.3.3.1. For each newly inserted alarm or newly changed alarm with a type change in the alarm list, if no display threads responsible for the alarm type of the alarm currently exist, then create a new display thread for the alarm type of the alarm. Then the alarm thread will print:
"First New *Display Thread(<thread_id>) Created at <creation_time>: <type time message>*".

A.3.3.2. For each newly inserted alarm or newly changed alarm with a type change in the alarm list, if all existing display threads responsible for the alarm type of the alarm have already been assigned two (2) alarms, then create a new display thread responsible for the alarm type of the alarm. Then the alarm thread will print:
"Additional New *Display Thread(<thread_id> Created at <creation_time>: <type time message>*".

A.3.3.3. For each newly inserted alarm or newly changed alarm with a type change in the alarm list, assign the alarm to any display thread responsible for the alarm type of the alarm that has fewer than two (2) alarms assigned to it. Then the alarm thread will print:
"*Alarm (<alarm-id>) Assigned to Display Thread(<thread_id>) at <assign_time>: <type time message>*".

A3.4. <u>The display threads in "New_alarm_mutex.c"</u>

Each display thread is responsible for displaying alarm messages for alarm with one alarm type and will do the following:

A.3.4.1. If the expiry time of an alarm assigned to the display thread in the alarm list has been reached, then the display thread will stop printing the message in that alarm. Then the display thread will print:

"*Alarm(<alarm_id>) Expired; Display Thread (<thread-id>) Stopped Printing Alarm Message at <stop_display_time>: <type time message>*".

A.3.4.2. **if an alarm assigned the display thread in the alarm list has been cancelled, then the display thread will stop printing the message in that alarm.** Then the display thread will print:
"*Alarm(<alarm_id>) Cancelled; Display Thread (<thread-id>) Stopped Printing Alarm Message at <stop_display_time>: <type time message>*".

A.3.4.3. **if the alarm type of an alarm assigned the display thread in the alarm list has been changed, then the display thread will stop printing the message in that alarm.** Then the display thread will print:
"*Alarm(<alarm_id>) Changed Type; Display Thread (<thread-id>) Stopped Printing Alarm Message at <stop_display_time>: <type time message>*".

A.3.4.4 If any display thread does not have any more alarm messages to display, then that display thread will terminate. Before terminating, that display thread will print:
"*Display Thread Terminated (<thread-id>) at <terminate_time>: <type time message>* ".

A.3.4.5. For each alarm with an alarm type which the display thread is responsible for and the alarm has been assigned by the alarm thread to that display thread, the display thread will periodically print, every five (5) seconds, the message in that alarm as follows:
"*Alarm(<alarm_id>) Message PERODICALLY PRINTED BY Display Thread (<thread-id>) at <periodic_print_time>: <type 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 the TA/marker is able to fully understand the design and implementation of your systems, and also fully understand your explanations regarding how the algorithms used in your programs actually work by reading your group's report.

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

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

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

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

(f) 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.

(g)
(g1) Your c program source file must be named: "**new_alarm_mutex.c**"

(g2) You must make sure that the TAs/markers will be able to successfully compile your program on the Red server system at York by using the following specific command:

cc new_alarm_mutex.c -D_POSIX_PTHREAD_SEMANTICS -lpthread

(g3) You must make sure that the TAs/markers, after compiling your program using the command in (g2), will be able to start to run your program on the Red server system at York by using the following specific command:

a.out

(g4) You must make sure that the TAs/markers, after compiling your program using the command in (g2) and using the command in (g3) to start running your program, your program will be able to handle any arbitrary sequence of alarm requests, including "Alarm requests", "Start_Alarm", "Change_Alarm", "Cancel_Alarm" and "View_Alarms" requests on the Red server system at York.

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

D. What to Hand In

D1. Each group is required to hand in an electronic copy of the following to eClass:

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.

D2. Each group is also required to use the utility "submit" to submit the electronic version of the above 3 files plus the "errors.h" file to the course directory
/cs/course/3221E/submit/a2
on the Red server.

(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 on the Red server.)

You may also submit your assignment through https://webapp.eecs.yorku.ca/submit

Important Warning:

Only submitting an electronic copy of your group’s assignment to eClass is not enough! If your group fails to use the utility "submit" to submit the electronic version of the above 3 files plus the “errors.h” file to the course directory /cs/course/3221E/submit/a2 on or before the due date, your group’s assignment will receive a grade of ‘F’.

E. Evaluation of the Assignment

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.

(f) The quality of the explanations/examples in your group’s report.

(g) How easy it is for the TA/marker to fully understand the design and implementation of your programs, and also fully understand your group’s explanations regarding how your group’s program actually work by reading your group’s 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.

(d) The quality of the code and the comments in the program.

(e) Whether the TA/marker will be able to compile, run your programs and test your program on the Red server system at York without any problems / issues.

F. Notes Regarding Academic Honesty

(a) Please note that this assignment must be completed independently by each group.

(b) Any form of collaboration or any form of collusion between each group and anyone else on this assignment is strictly forbidden.

(c) Please read very carefully York University Policies and the EECS Department Policies regarding academic dishonesty and plagiarism at:
http://www.cse.yorku.ca/admin/coscOnAcadHonesty.html

(d) Please be very vigilant about preventing academic dishonesty and plagiarism when completing this assignment, as the consequences of violating York University Policies and the EECS Department Policies regarding academic dishonesty and plagiarism can be very serious.

(e) Please also note that, when each group hands in this assignment, each group is also required to complete and hand in the Lassonde "Group Assignment Checklist", which is posted on eClass.

G. 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.

Please also note that *the due date of this assignment,* ***Monday November 4, 2024, 23:59*** *cannot be extended*. So please plan carefully in advance in order to make sure that you will be able to complete this assignment before the posted due date.