EECS 3221 Section E Fall 2024
Assignment 3

POSIX Threads, Semaphores, Readers-Writers Problem

Due Date: Last Day to Hand in Term Work, that is, Tuesday December 3, 2024, 23:59.

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 David R. Butenhof (This book is currently on reserve at Steacie Science Library, Call Number QA 76.76 T55 B88 1997).

You are also required to read Chapter 7, p.301-302, of the course textbook, "Operating System Concepts," 10th Edition, by A. Silberschatz et al., on how to use POSIX unnamed semaphores.

A2. You are required to download the program "alarm_cond.c" and the file "errors.h" and "README" from the directory /cs/course/3221E/assign3, and then try to compile and execute this program by following the instructions in the "README" file. (This program is explained in pages 82-88 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.)

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

- There are 4 different types of threads in "new_alarm_mutex.c":
(a) A main thread;
(b) An "alarm group_display creation_thread";
(c) An "alarm group_display removal thread";
(d) "Display alarm threads".

- The alarm list must be the only data structure that is shared between the above 4 different types of threads in "new_alarm_mutex.c".

A3.1 Six types of "Alarm requests" in "New_Alarm_Cond.c"

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

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

Alarm> Start_Alarm(Alarm_ID): Group(Group_ID) Time Message

- "Start_Alarm" is a keyword.
- "Alarm_ID" is a positive integer.
- "Group" is a keyword.
- "Group_ID" is a positive integer.
- ***"Time" in the program "New_Alarm_Cond.c" has a same syntax but a different meaning compared within the program "alarm_mutex.c":***
  "Time" in the program "New_Alarm_Cond.c" means that "Message" should be displayed once every Time seconds.
- "Message" has the same meaning and syntax as in the program "alarm_mutex.c".

For example:
Alarm> Start_Alarm(2345): Group(13) 5 Will meet you at Grandma's house at 6 pm

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

Alarm> Change_Alarm(Alarm_ID): Group(Group_ID) Time Message

- "Change_Alarm" is a keyword.
- "Alarm_ID" is a positive integer.
- "Group" is a keyword.
- "Group_ID" is a positive integer.
- ***"Time" in the program "New_Alarm_Cond.c" has a same syntax but a different meaning compared within the program "alarm_mutex.c":***
  "Time" in the program "New_Alarm_Cond.c" means that "Message" should be displayed once every Time seconds.
- "Message" has the same meaning and syntax as in the program "alarm_mutex.c".

For example:
Alarm> Change_Alarm(2345): Group(21) 8 Will meet you at Grandma's house later at 8 pm

(c) "Cancel_Alarm" requests with the following format:

Alarm> Cancel_Alarm(Alarm_ID)

- "Cancel_Alarm" is a keyword.
- "Alarm_ID" is a positive integer.

For example:

Alarm> Cancel_Alarm(2345)

(d) "Suspend_Alarms" requests with the following format:

Alarm> Suspend_Alarm(Alarm_ID)

- "Suspend_Alarm" is a keyword.
- "Alarm_ID" is a positive integer.

For example:
Alarm> Suspend_Alarm(2345)

(e) "Reactivate_Alarms" requests with the following format:

Alarm> Reactivate_Alarm(Alarm_ID)

- "Reactivate_Alarm" is a keyword.
- "Alarm_ID" is a positive integer.

For example:
Alarm> Reactivate_Alarm(2345)

(f) "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 six 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_Cond.c"

The main thread in "New_alarm_mutex.c" first creates an "alarm group_display creation_thread" and an "alarm group_display removal thread", which are described in section A.3.3 and section A.3.4 respectively further below.

The main thread in "New_Alarm_Cond.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 request with the specified Alarm_ID into the alarm list, *in which all the alarm requests 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>: Group(<group_id>) <time message>".

A.3.2.2. **For each Change_Alarm request** in the alarm list, the main thread will use the specified Group_ID, Time and Message values in the Change_Alarm request to **replace** the Group_ID, 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 <alarm_change_time>: Group(<group_id>) <time message>".

A3.2.3. **For each 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>) Canceled at <cancel_time>: <time message>".

A3.2.4. **For each Suspend_Alarm request** received, the main thread will change the status of the alarm with the specified Alarm_Id in the alarm list from *"active"* to *"suspended*"; then the main thread will print:
Alarm(<alarm_id>) Suspended at <suspend_time>: <time message>".

Note: When an alarm is first inserted into the alarm list, the status of that alarm is set to "*active"*.

A3.2.5. **For each Reactivate_Alarm request** received, the main thread will the main thread will change the status of the alarm with the specified Alarm_Id in the alarm list from *"suspended"* to *"active*";; then the main thread will print:
Alarm(<alarm_id>) Reactivated at <reactivate_time>: <time message>".

A3.2.6. **For each View_Alarms request** received, the main thread will print out the following:
- A list of all the current existing display alarm threads, together with the alarms that each display alarm thread has been assigned to, in the following format:
"View Alarms at <view_time>: <time>:
1. Display Thread <thread-id> Assigned:
  1a. Alarm(<alarm_id>): Created at <create_time> Assigned at <assign_time> <time message> Status<status>
  1b. Alarm(<alarm_id>): Created at <create_time> Assigned at <assign_time> <time message> Status<status> **(if a second alarm is a assigned)**
  1c …
2. Display Thread <thread-id> Assigned:

2a. Alarm(<alarm_id>): Created at <create_time> Assigned at <assign_time> <time message> Status<status>
2b. Alarm(<alarm_id>): Created at <create_time> Assigned at <assign_time> <time message> Status<status> **(if a second alarm is assigned)**
2c …
.
Note that above, and in each of the messages printed by the various threads below, <thread-id> is the thread identifier; <group_id> is the group identifier; <insert_time>, <create_time>, <assign_time>, <cancel_time", <suspend_time>, <reactivate_time>, <cancel_remove_time>, <change_time>, <display_change_time>, <print_time>, <remove_time>, <alarm_change_time>, <stopped_print_time>, ,<old_group_change_time>, <new_group_change_time>, <message_change-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; <group_id>, <time message> correspond to "Group_ID", Time" and "Message" as specified in A3.1 (a), (b) above.

A3.3. The alarm group display creation thread in "New_Alarm_Cond.c"

The alarm group display creation thread is responsible for creating display alarm threads, such that each display alarm thread will only display/print messages for one and only one group of alarms which have a same particular Group_ID as follows.

The alarm group display creation thread always waits until the alarm list has been changed, then for each alarm in the alarm list, the alarm group display creation thread will do the following:

If there does not yet exist a display alarm thread which is responsible for displaying/printing messages for alarms with the particular Group_ID of that alarm,

Then: (a) create a new display alarm thread which will be responsible for displaying/printing messages for alarms with the particular Group_ID of that alarm and assign that alarm to the newly created display alarm thread. Then the alarm group display creation thread will print:
"*Alarm Group Display Creation Thread Created New Display Alarm Thread <thread-id> For Alarm(<alarm_id> at <create_time>: Group(<group_id>) <time message>*".

Else: (b) assign that alarm to an existing display alarm thread which is responsible for displaying/printing messages for alarms with the particular Group_ID of that alarm. Then the alarm group display creation thread will print:
"*Alarm Thread Display Alarm Thread <thread-id> Assigned to Display Alarm(<alarm_id> at <assign_time>: Group(<group_id>) < time message>*".

A3.4. The alarm group display removal thread in "New_Alarm_Cond.c"

The alarm group display removal thread always waits until the alarm list has been changed, then for each display alarm thread responsible for displaying/printing

messages for a particular group of alarms, if the alarm group display removal thread detects that there are no more alarms belonging that particular group in the alarm list, then the alarm group display removal thread will terminate that display alarm thread.
Then the alarm removal thread will print**:**
"No More Alarms in Group(<group_id> Alarm Removal Thread Has Removed Display Alarm Thread <thread-id> at <remove_time>: Group(<group_id>) <time message>".

A3.5. The display alarm_threads in "New_Alarm_Mutex.c"

A3.5.1. After each display alarm thread in "New_Alarm_Mutex.c" has been created by the alarm display creation thread, for each alarm with the same particular Group ID that has been assigned to that display alarm thread, **that display alarm thread will periodically print, every "Time" seconds**, the following:

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

A.3.5.2. For each alarm, **if that alarm has been removed from the alarm list by the alarm removal thread**, then **the display alarm thread responsible for displaying/printing messages for the group of that alarm will stop printing the message in that alarm.** Then the display alarm thread will print:
"Display Thread <thread-id> Has Stopped Printing Message of Alarm(<alarm_id>at <stopped_print_time>: Group(<group_id>) <time message>".

A.3.5.3. For each alarm in the alarm list, **if the Group ID of that alarm has been changed**, then **the display alarm thread responsible for displaying/printing messages for the *previous* group of that alarm will stop printing the message in that alarm** and will print:
"Display Thread <thread-id> Has Stopped Printing Message of Alarm(<alarm_id>at <old_group_change_time>: Changed Group(<group_id>) <time message>".
At the same time, **the display alarm thread responsible for displaying/printing messages for the *new* group of that alarm will start periodically printing, every "Time" seconds, the message in that newly changed alarm** and will 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>".

A.3.5.4. For each newly changed alarm in the alarm list, **if the Group ID of that alarm has NOT been changed but the message in that alarm has been changed, then the display alarm thread responsible for displaying/printing messages for alarms in the particular Group ID of that alarm will start periodically printing, every "Time" 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 Alarm(<alarm_id> at <message_change_time>: Group(<group_id>) < time message>".

A.3.5.5. For each alarm in the alarm list that has been assigned to this display alarm thread, **if that alarm has been suspended, then** **the display alarm thread will stop printing the message in that alarm.**

A3.5.6. For each alarm in the alarm list that has been assigned to this display alarm thread, **if that alarm has been reactivated then** **the display alarm thread will continue printing the message in that alarm.**

A3.5. Treating synchronization of the thread accesses to the alarm list as solving a "Readers-Writers" problem in "New_Alarm_Cond.c"

A.3.5.1. *You are required to treat synchronization of the thread accesses to the shared data – the alarm list, as solving a "Readers-Writers" problem in "New_Alarm_Cond.c".* Any thread that needs to modify the alarm list, should be treated as a "writer process - only one writer process should be able to modify the alarm list at a time. Any thread that only needs to read information from the alarm list, should be treated as a reader process - any number of reader processes should be able to read information from the alarm list simultaneously.

A3.5.2. *You are required to use POSIX unnamed semaphores, described in Chapter 7, p.301-302, of the course textbook, "Operating System Concepts," 10th Edition, to synchronize thread accesses to the alarm list.*

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 Prism 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_cond.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_cond.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 correctly handle any arbitrary sequence of alarm requests that includes any number of the six types of alarm requests "Start_Alarm", "Change_Alarm", "Cancel_Alarm" and "Suspend_Alarm", "Activate_Alarm", and "View_Alarms"specified in section 3.1 on the Red server system at York.

(h) Your group's report must clearly identify which sources of information you have used in which components of this assignment. Your group's report must make sure that the source of each portion of the program algorithms/code, and explanations/examples in your group's assignment is clearly identified and attributed.

(i) Your group's report must make sure that your group clearly identifies which particular element(s) of the program algorithms/code, and explanations/examples in your group's assignment are your group's own work. If your group has provided some algorithms/code or written explanation/example that cannot be found anywhere else, your group should explicitly identify it as such, by writing in brackets that this is your group's own algorithm/code, explanation/example.

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

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

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 submit the electronic version of the above 3 files plus the "errors.h" file to the course directory /cs/course/3221E/submit/a3
on the Red server by clicking on the link https://webapp.eecs.yorku.ca/submit, then following the instructions.

(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.)

Important Warning:

Only submitting an electronic copy of your group's assignment to eClass is not enough! If your group fails 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 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 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) How well your programs satisfy the Additional Requirements (a), (f), (h), (i) in section C above.

(c) The originality and quality of the algorithms/code, explanations/examples in your group's assignment.

(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) How well your programs satisfy the Additional Requirements (b), (c), (d), (e), (g) in section C above.

(d) The quality of the program code and the comments that explain the program code that is your group's own program code.
Please note that, a group's assignment program code that copies code from a textbook or an online source, will receive a low grade compared with a group's assignment's program code that is a group's own program code.

(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,* **Tuesday December 3, 2024, 23:59** *falls on the Last Day to Hand In Term Work* according to the University Regulations. Thus it will not be possible to postpone the due date of this assignment. 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.