EECS 3221 Section Z Winter 2024
Assignment 3

POSIX Threads, Semaphores, Producer-Consumer-Shared-Circular-Buffer and Readers-Writers Problem

Due Date: Last Day to Hand in Term Work, that is, Monday April 8, 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).

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

In addition to the main thread, the program "New_Alarm_Cond.c" should also have an "alarm thread", a "consumer thread", and zero or more "periodic display threads". In addition to the "Alarm List", which is the only data structure that is shared between the main thread and the alarm thread, the program "New_Alarm_Cond.c" should also have two additional shared data structures: a "Circular_Buffer" which is the only data structure that is shared between the alarm thread and the consumer thread; and an "Alarm_Display_List" which is the only data structure that is shared between the consumer thread and the periodic display threads.

The main thread should not share any data structure with the consumer thread and the periodic display threads. The alarm thread also should not share any data structure with the periodic display threads.

Furthermore:

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

Three types of "Alarm requests", "Start_Alarm", "Change_Alarm" and "Cancel_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): Time Message

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

For example:
Alarm> Start_Alarm(2345): 4 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” is a positive integer.
- “Message” has the same meaning and syntax as in the program “alarm_mutex.c”.

For example:
Alarm> Change_Alarm(2345): 5 Will meet you at Grandma’s house later at 8 pm

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

Alarm> Cancel_Alarm(Alarm_ID): Time

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

For example:
Alarm> Cancel_Alarm(2345) 5

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

Below “Time value” means the positive integer value of “Time” given in the “Start_Alarm”, “Change_Alarm” or “Cancel_Alarm” requests above.

A3.2. <u>The main thread in “New_Alarm_Cond.c”</u>

The main thread is responsible for creating one alarm thread, and one consumer thread.

If there exist any errors in any alarm request, then the main thread in "New_Alarm_Cond.c" will reject the alarm request with an error message. If a Message exceeds 128 characters, it will be truncated to 128 characters.

For each Start_Alarm, Change_Alarm, or Cancel_Alarm request received without errors, the main thread will **insert** the alarm with the specified Alarm_ID into the Alarm List, *in which all the alarms requests are placed in the order of their Time values.* Then the main thread will print:
"*Main Thread has Inserted Alarm_Request_Type <alarm-request-type> Request(<alarm_id>) at <insert_time>: Time = <time> Message = <message> into Alarm List*"

Above *<alarm-request-type> can be either "Start_Alarm", or "Change_Alarm", or "Cancel_Alarm".*

A3.3. The alarm thread in "New_Alarm_Cond.c"

A.3.3.1. The alarm_thread in "New_Alarm_Cond.c" will wait until the Alarm List has been changed by the main thread, then the alarm thread will examine the most recent alarm request that was inserted into the Alarm List.

A.3.3.2. If the most recent alarm request is a Cancel_Alarm request, then the alarm thread will remove all alarm requests with the specified Alarm_ID from the Alarm List. Then the alarm thread will print:
"Alarm Thread<thread-id> Has Cancelled and Removed All Alarm Requests With Alarm ID <alarm_id> from Alarm List at <remove_time>"

A.3.3.3. If the most recent alarm request is a Change_Alarm request, then the alarm thread will first remove all alarm requests with the specified Alarm_ID except the most recent Change_Alarm request from the Alarm List. Then the alarm thread will print:
"Alarm Thread<thread-id> at <change_time> Has Removed All Alarm Requests With Alarm ID <alarm_id> From Alarm List Except The Most Recent Change Alarm Request(<alarm_id>) Time = <time> Message = <message>"

A.3.3.4. If the most recent alarm request is a Start_Alarm, or Change_Alarm, request, and the alarm request has a new Time value such that none of the previous alarm requests in the Alarm List has a same Time value, then the alarm thread will create a new periodic display thread to display messages in all alarm requests with the new Time value.
Then the alarm thread will print:
"*Alarm Thread Created New Periodic display thread <thread-id> For Alarm(<alarm_id>) at <create_time>: For New Time Value = <time> Message = <message>*".

A.3.3.5. If the most recent alarm request is a Start_Alarm, Change_Alarm, or Cancel_Alarm request, then the alarm_thread will insert the most recent alarm request

into a ***Circular_Buffer*** which it shares with a *consumer thread* specified in the next section.
Then the alarm thread will print:
"*Alarm Thread has Inserted Alarm_Request_Type <alarm-request-type> Request(<alarm_id>) at <insert_time>: Time = <time> Message = <message> into Circular_Buffer Index: <circular-buffer-index>"*.

Above *<circular-buffer-index> is the Circular-Buffer array index into which the alarm thread inserted the alarm request. <alarm-request-type> can be either "Start_Alarm", or "Change_Alarm", or "Cancel_Alarm".*

The shared Circular_Buffer can contain a maximum of 4 items.

A.3.3.6. Then the alarm thread will print all the alarm requests currently in the Alarm List in the order of their Time values.

A3.4. <u>The "consumer_thread" in "New_Alarm_Cond.c"</u>

As mentioned above, the main thread in "New_Alarm_Cond.c" will also create a "consumer_thread", described below.

The consumer_thread is responsible for retrieving alarm requests one-by-one from the shared Circular_Buffer specified in the previous section.

A.3.4.1. For each alarm request retrieved from the circular buffer, the alarm thread will print:
"*Consumer Thread has Retrieved Alarm_Request_Type <alarm-request-type>Request (<alarm_id>) at <retrieve_time>: Time = <time> Message = <message> from Circular_Buffer Index: <circular-buffer-index>"*.

Above *<circular-buffer-index> is the Circular-Buffer array index from which the consumer thread retrieved the alarm request. <alarm-request-type> can be either "Start_Alarm", or "Change_Alarm", or "Cancel_Alarm".*

A.3.4.2. If the retrieved alarm request is a Start_Alarm request, then the consumer thread will insert the Start_Alarm request into the Alarm Display List, *in which all the outstanding alarm requests are placed in the order of their Time values.*
Then the consumer thread will print:
"*Consumer Thread has Inserted Alarm_Request_Type <alarm-request-type> Request(<alarm_id>) at <insert_time>: Time = <time> Message = <message> into Alarm Display List."*

A.3.4.3. If the retrieved alarm request is a Change_Alarm request, then the consumer thread will first remove all previous alarm requests with the specified Alarm_ID from the

Alarm Display List; then the consumer thread will insert the retrieved Change_Alarm request into the Alarm Display List.
Then the consumer thread will print:
"Consumer Thread<thread-id> at <change_time> has Removed All Previous Alarm Requests With Alarm ID <alarm_id> From Alarm Display List and Has Inserted Retrieved Change Alarm Request(<alarm_id>) Time = <time> Message = <message>" into Alarm Display List."

A.3.4.4. If the retrieved alarm request is a Cancel_Alarm request, then the consumer thread will remove all alarm requests with the specified Alarm_ID from the Alarm Display List. Then the alarm thread will print:
"Consumer Thread<thread-id> Has Cancelled and Removed All Alarm Requests With Alarm ID (<alarm_id>) from Alarm Display List at <remove_time>"

A.3.4.5. Then the alarm thread will print all the alarm requests currently in the Alarm Display List in the order of their Time values.

A3.5. <u>The periodic display threads in "New_Alarm_Mutex.c"</u>

A3.5.1. For all alarms with a same particular Time value, one of the periodic display threads will be responsible for periodically printing the messages of all those alarms with that particular Time value, and will <u>***periodically print, every Time seconds***, the following</u>:

"***ALARM MESSAGE (<alarm_id>) PRINTED BY ALARM DISPLAY THREAD <thread-id>at <print_time>: TIME = <time> MESSAGE = <message>*** ".

A.3.5.2. For each alarm, <u>**if that alarm has been removed from the Alarm Display List by the consumer thread, then**</u> <u>**the periodic display thread will stop printing the message in the removed alarm.**</u> Then the periodic display thread will print:
"***Display Thread <thread-id> Has Stopped Printing Message of Alarm(<alarm_id>) at <stopped_print_time>: Time = <time> Message = <message>***".

A.3.5.3. For each alarm in the Alarm Display List, <u>**if the Time value of that alarm has been changed by the consumer thread**, **then**</u> <u>**the periodic display thread responsible for displaying/printing messages for all alarms with the same Time value as the *previous* Time value of the changed alarm will stop printing the message in the changed alarm**</u> and will print:
"***Display Thread <thread-id> Has Stopped Printing Message of Alarm(<alarm_id>) at <old-change-time>: Old Changed Time = <time> Message = <message>***".

A.3.5.4. For each alarm in the Alarm Display List, <u>**if the Time value of that alarm has been changed by the consumer thread, then another periodic display thread responsible for displaying/printing messages for all alarms with the *new* Time value will print**</u>:

“***Display Thread <thread-id> Has Taken Over Printing Message of Alarm(<alarm_id>) at <new-change_time>: New Changed Time = <time> Message = <message>***”.
Then, as described in A3.5.1 above, this alarm display thread will start to periodically print, every Time seconds, the message in the changed alarm.

A.3.5.5. For each newly changed alarm in the Alarm Display List, **if the Time value of that alarm has NOT been changed but the Message in that alarm has been changed, then the periodic display thread responsible for displaying/printing messages for all alarms with the particular Time value of that alarm will print**:
“***Display Thread <thread-id> Starting to Print Changed Message Alarm(<alarm_id>) at <message_change_time>: Time = < time> Message = <message>***”.

A.3.5.6. For each periodic display thread **responsible for displaying/printing messages for all alarms with a particular Time value, if that display alarm thead detects that there are no more alarms with that particular Time value in the Alarm Display List, then that periodic display thread will first print:**
“***No More Alarms With Time = <time> Display Thread <thread-id> exiting at <exit_time>***”.
Then the periodic display thread will exit (terminate).

A.3.5.7. **Periodic display threads should not change any information in the Alarm Display List.**

Note that in each of the messages printed by the various threads above, <thread-id> is the thread identifier; <time> is the Time value; <message> is the Message string; <insert_time>, <create_time>, <cancel_time>, <change_time>, <retrieve_time>, <print_time>, <stopped_print_time>, <exit_time>, <old_change_time>, <new_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; Time = <time> Message = <message> correspond to the positive integer value of Time” and the character string in “Message” as specified in A3.1 (a), (b), (c) above.

A3.5. Treating synchronization of the thread accesses to the Alarm Display 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 Display List, as solving a “Readers-Writers” problem in “New_Alarm_Cond.c”*. Any thread that needs to modify the Alarm Display List, should be treated as a “writer process - only one writer process should be able to modify the Alarm Display List at a time. Any thread that only needs to read information from the Alarm Display List, should be treated as a reader process - any number of reader processes should be able to read information from the Alarm Display 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 Display List.*

B. <u>Platform on Which The Programs Are to be Implemented</u>

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. <u>Additional Requirements</u>

(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) You must make sure that the TAs/markers will be able to compile, run and test your programs on the Red server system at York without any difficulty by following the instructions that you provide in your "README" file.

(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 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 code, and explanations/examples in your group's assignment are your group's own work. If your group has provided some 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 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>

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 have 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, run and test your programs, and which also provides the specific commands that the TA/marker can use to compile, run and test your programs on the Red server system at York..

Each group is also required to submit the electronic version of the above 5 files plus the file "errors.h" to the course directory /cs/course/3221Z/submit/a3
on the Red server.

You may 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 submit the electronic version of the above 4 files to the course directory /cs/course/3221Z/submit/a3 on or before the due date, your group's assignment will receive a grade of 'F'.

(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.)

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 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 that are your group's own work..

(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 the algorithms used in your group's programs 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 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 TAs/markers will be able to compile, run your programs and test your information systems on the Red server system at York without any difficulty by following the instructions that you provide in your "README" file.

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 April 8, 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.