EECS 3221 Section E Fall 2019
Assignment 3

POSIX Threads, Semaphores, Readers-Writers Problem

Due Date: Last Day to Hand in Term Work, that is, Wednesday December 4, 2019, 23:59.

A. <u>Description of the Assignment</u>

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 and alarm_thread in the program "alarm_cond.c", the program "New_Alarm_Cond.c" will have additional "periodic_display_threads". Furthermore:

A3.1 <u>Two types of "Alarm requests" in "New_Alarm_Cond.c"</u>

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

(a) Alarm requests which include, as a second parameter, a Message Number, with the following format:

Alarm> Time Message(Message_Number) Message

- "Time" has the same meaning and syntax as in the program "alarm_cond.c".
- "Message_Number" is a positive integer.
- "Message" has the same meaning and syntax as in the program "alarm_cond.c".

For example:
Alarm> 5 Message(2) Buy groceries at Metro

(b) Alarm requests prefixed with the key word "Cancel:", with the following format:

Alarm> Cancel: Message(Message_Number)

- "Cancel:" is a keyword.
- "Message_Number" is a positive integer.

For example:
Alarm> Cancel: Message(2)

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. <u>The main thread in "New_Alarm_Cond.c"</u>

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

A.3.2.1. If there does not exist any alarm request with the same Message Number in an alarm list, then the main thread will insert the alarm request into the alarm list, *in which all the outstanding alarm requests are placed in the order of their Message Numbers.*.

A.3.2.2. If there exists an alarm request with the same Message Number in the alarm list, then the main thread will replace that alarm request with the most recent alarm request in the alarm list.

A3.3. <u>The alarm_thread in "New_Alarm_Cond.c"</u>

The alarm_thread in "New_Alarm_Cond.c" checks alarm requests in the alarm list whenever the alarm list has been changed.

A.3.3.1. On finding a new alarm request that is NOT prefixed with the keyword "Cancel:" in the alarm list,, the alarm_thread will immediately create a *periodic_display_thread* specified in the next section, and also immediately print "DISPLAY THREAD CREATED FOR: Message(<Message_Number>) <Message>".
For example:
DISPLAY THREAD CREATED FOR: Message(2) Buy groceries at Metro

A.3.3.2. On finding a alarm request prefixed with the keyword “Cancel:” in the alarm list, the alarm_thread will *immediately* print
“CANCEL: Message(<Message_Number>) <Message>”. For example:
    CANCEL: Message(2) Buy groceries at Metro
Then the alarm_thread will remove the data corresponding to the alarm request with the corresponding Message Number from alarm list.

A3.4. <u>The “periodic_display_threads” in “New_Alarm_Cond.c”</u>

A3.4.1. Each periodic_display_thread is responsible for periodically looking up an alarm request with a specific Message Number in the alarm list, then printing, every Time seconds, where Time is the first parameter in an alarm request as described in A3.1 (a) above:
“Message(<Message_Number>) <Message>”.
For example:
  Message(2) Buy groceries at Metro

A3.4.2. If a periodic_display_thread, when periodically looking up an alarm request with a specific Message Number in the alarm list, finds that an alarm request with a specific Message Number in the alarm list has been changed, then it will first print:

“MESSAGE CHANGED: Message(<Message_Number>) <Message>”

then start printing, every Time seconds, where Time is the first parameter in the modified alarm request:
“Message(<Message_Number>) <Message>”.
For example:
  Message(2) Buy groceries at Loblaws instead

A3.4.3. If a periodic_display_thread finds that the alarm request with a specific Message Number has been removed from the alarm list by the alarm thread because that alarm request has been cancelled, then that periodic_display_thread will first print:
“DISPLAY THREAD EXITING: Message(<Message_Number>) ”.
For example:
  DISPLAY THREAD EXITING: Message(2)
Then the periodic_display_thread will exit.

A3.5. <u>Treating synchronization of the thread accesses to the alarm list as solving a “Readers-Writers” problem in “New_Alarm_Cond.c”</u>

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. <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 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 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 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/3221E/submit/a3

(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. Evaluation of the Assignment

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.

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.

Please also note that *the due date of this assignment, Wednesday December 4, 2019, 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.