

EECS 3221 Section M Winter 2020
Assignment 3

POSIX Threads, Semaphores, Readers-Writers Problem

Due Date: Last Day to Hand in Term Work, that is, Monday April 6, 2020, 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/3221M/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".

A3.1 Two 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" 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): Group(13) 50 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” 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): Group(21) 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 Cond.c”

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 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>: 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>”***.

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>, <remove_time>, <alarm_change_time>, <stopped_print_time>, <exit_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.

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 removal thread in “New Alarm Cond.c”

For each alarm in the alarm list, **if the expiry time of that alarm has been reached, then the alarm removal thread will remove that alarm from the alarm list.** Then the alarm removal thread will print:

“Alarm Removal Thread Has Removed Alarm(<alarm_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 5 seconds, the following:

“Alarm (<alarm_id>) Printed by Alarm Display 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 5 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 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 Alarm(<alarm_id> at <message_change_time>: Group(<group_id>) <time message>”.

A.3.5.5. For each display alarm thread responsible for displaying/printing messages for a particular group of alarms, if that display alarm thread detects that there are no more alarms belonging that particular group in the alarm list, then that display alarm thread will first print:

“No More Alarms in Group(<group_id>): Display Thread <thread-id> exiting at <exit_time>”.

Then the display alarm thread will exit (terminate).

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 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. What to Hand In

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, Monday April 6, 2020, 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.