

**EECS3221 Winter 2018
Assignment 3**

POSIX Threads, Semaphores, Readers-Writers Problem

Due Date: Last Day to Hand in Term Work, that is, Thursday April 5, 2018, 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 5, pages 237-238 of the course textbook, "Operating System Concepts," 9th 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/3221/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 Three types of "Alarm requests" in "New_Alarm_Cond.c"

Three types of "Alarm requests", "Type A", "Type B", and "Type C" Alarm requests, are recognized and handled by "New_Alarm_Cond.c", as follows:

(a) "Type A" Alarm requests which include, as a first parameter, a Time value; as a second parameter, a Message Type; as a third parameter, a Message Number; and, as a fourth parameter, a Message; with the following format:

Alarm> Time Message(Message_Type, Message_Number) Message

- "Time" has the same meaning and syntax as in the program "alarm_cond.c".
- "Message_Type" and "Message_Number" are both positive integers.
- "Message" has the same meaning and syntax as in the program "alarm_cond.c".

For example:

Alarm> 5 Message(2, 13) Visit Grandma on way back home

(b) “Type B” Alarm requests prefixed with the key word “Create_Thread:”, with the following format:

Alarm> Create_Thread: MessageType(Message_Type)

- “Create_Thread:” is a keyword.
- “Message_Type” is a positive integer.

For example:

Alarm> Create_Thread: MessageType(2)

(c) “Type C” 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(13)

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.

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 Type A alarm request received, if there does not exist any Type A 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*, and the main thread will print: **“Type A Alarm Request With Message Number (Message_Number) Inserted Into Alarm List at <time>: <alarm_request>”**.

Note that above, and in each of the messages printed by the various threads below, <time> is the actual time at which the alarm was received; <time> is expressed as the

number of seconds from the Unix Epoch Jan 1 1970 00:00; <Message_Number> and <alarm_request> are as specified in A3.1 (a) or (b).

A.3.2.2. For each Type A alarm request received, 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, and the main thread will print:

“Type A Replacement Alarm Request With Message Number (Message_Number) Inserted Into Alarm List at <time>: <alarm_request>”.

Note that the alarm list cannot have more than one alarm request with the same Message Number. An implication of A.3.2.2 is that this may result in a change in the Message Type for an alarm request with a specified Message Number, and this could also affect other operations below.

A.3.2.3. For each Type B alarm request received, if there does not exist any Type A alarm request with the specified Message Type in an alarm list, then the main thread will print the error message:

“Type B Alarm Request Error: No Alarm Request With Message Type (Message_Type).!”

A.3.2.4. For each Type B alarm request received, if there exists a Type B alarm request with the same Message Number in the alarm list, then the main thread will print the error message:

“Error: More Than One Type B Alarm Request With Message Type (Message_Type).!”

A.3.2.5. For each Type B alarm request received, if there exists a Type A alarm request with the same Message Type and there does not exist a Type B alarm request with the same Message Type in the alarm list, then the main thread will insert the Type B alarm request into the alarm list, and the main thread will print:

“Type B Create Thread Alarm Request For Message Type (Message_Type) Inserted Into Alarm List at <time>!”.

A.3.2.6. For each Type C alarm request received, if there does not exist any Type A alarm request with the same Message Number in an alarm list, then the main thread will print the error message:

“Error: No Alarm Request With Message Number (Message_Number) to Cancel!”

A.3.2.7. For each Type C alarm request received, if there exists a Type C alarm request with the same Message Number in the alarm list, then the main thread will print the error message:

“Error: More Than One Request to Cancel Alarm Request With Message Number (Message_Number).!”

A.3.2.8. For each Type C alarm request received, if there exists a Type A alarm request with the same Message Number and there does not exist a Type C alarm request with the

same Message Number in the 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*, and the main thread will print:

“Type C Cancel Alarm Request With Message Number (Message_Number) Inserted Into Alarm List at <time>: <alarm_request>”..

A3.3. The alarm thread in “New Alarm Cond.c”

The alarm_thread in “New_Alarm_Cond.c” checks alarm requests in the alarm list **whenever a Type A or Type B or Type C alarm request has been inserted into the alarm list.**

A.3.3.1. On finding a new Type A alarm request in the alarm list, the alarm_thread will do the following:

If a periodic_display_thread responsible for displaying messages with a specific Message_Type had been previously created, *but there does not exist any more alarm requests in the alarm list with that Message Type as a result of the operation in A.3.2.2 above which may change the Message Type of an alarm request that was previously inserted into the alarm list*, then terminate the periodic_display_thread responsible for displaying messages with the specified Message_Type, then print: ***“Type A Alarm Request Processed at <time>: Periodic Display Thread For Message Type (Message_Type) Terminated: No more Alarm Requests For Message Type (Message_Type).”***

A.3.3.2. On finding a new Type B alarm request in the alarm list, the alarm_thread will immediately create a *periodic_display_thread* (specified in the next section) *responsible for displaying messages with the specified Message_Type*, then print: ***“Type B Alarm Request Processed at <time>: New Periodic Display Thread For Message Type (Message_Type) Created.”***

A.3.3.3. On finding a new Type C alarm request in the alarm list, the alarm_thread will do the following:

A3.3.3 (a) Remove the Type A alarm request with the specified Message_Number from the alarm list, then print: ***“Type C Alarm Request Processed at <time>: Alarm Request With Message Number (Message_Number) Removed”.***

A3.3.3 (b) If there are no more alarm requests in the alarm list with a Message_Type that is the same as the Message Type of the Type A alarm request that was just removed from the alarm list as a result of operation A.3.3.3 (a), then *terminate the periodic_display_thread responsible for displaying messages with the specified Message_Type (if a periodic_display_thread responsible for displaying messages with the specified Message_Type has been created)*, then print: ***“Type C Alarm Request Processed at <time>: Periodic Display Thread For Message Type (Message_Type) Terminated: No more Alarm Requests For Message Type (Message_Type).”***

A3.4. The “periodic_display_threads” in “New_Alarm_Cond.c”

A3.4.1. Each periodic_display_thread is responsible for periodically looking up a Type A alarm request with a specific Message Type 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:

“Alarm With Message Type (Message_Type) and Message Number (Message_Number) Displayed at <time>: <alarm_request>”.

A3.4.2. If a periodic_display_thread, when periodically looking up a Type A alarm request with a specific Message Type in the alarm list, finds that a Type A alarm request which previously had that specific Message Type in the alarm list has changed to a different Message Type as a result of operation A3.2.2, then it will print:

“Stopped Displaying Replaced Alarm With Message Type (Message_Type) at <time>: <alarm_request>”;

Then it would stop perform the operation in A.3.4.1. for the alarm request which now has a different Message Type.

A3.5. Synchronization of thread accesses to the alarm list, by treating the threads as “readers” and “writers” in “New_Alarm_Cond.c”

A3.5.1. *You are required to synchronize thread accesses to the shared data – the alarm list, by treating the threads as “readers” and “writers”, and implementing a solution to the “Readers-Writers” problem that was described in Chapter 5, pages 220-222 of the course textbook, “Operating System Concepts,” 9th Edition, so that:*

A3.5.1 (a) 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;

A3.5.1 (b) 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 5, pages 237-238 of the course textbook, “Operating System Concepts,” 9th 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 has 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/3221/submit/a2

(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.)

Important Warning:

Only submitting an electronic copy of your assignment is not enough! If you fail to submit a hard copy of your assignment on or before the due date, your assignment will receive a grade of 'F', regardless of whether an electronic copy of the assignment was submitted 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, Thursday April 5, 2018, 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.