EECS3221 Section E

Operating System Fundamentals

Fall 2023

Assignment 3 Posix Threads

Team Members:

John Yacoub (217988908)

Yvan Semana (217099425)

Sathira Williams (218131938)

Mikhail Malinovskiy(218759977)

December 5 2023

Professor Jia Xu

Table of Contents

Abstract

This assignment requires the implementation of an alarm program to satisfy the requirements, using POSIX Threads, and Semaphores to solve the Readers-Writers problem.

1. Implementation Requirements

1.1 - File Creation

Make the changes and additions to the file of alarm_cond.c to a new file called New_Alarm_Cond.c.

1.2 - Alarm Requests

There are two type of alarm requests made by New_Alarm_Cond.c:

1. **Command**: "*Start_Alarm(Alarm_ID): Group(Group_ID) Time Message*"
    a. "Start_Alarm" is the command.
    b. "Alarm_ID" is a positive int.
    c. "Group" is a keyword.
    d. "Group_ID" is a positive int, to determine the group.
    e. "Time" is the amount of time till the alarm triggers.
    f. "Message" is the message displayed at the alarm's trigger.

    Start_Alarm: For each request received the main thread will **insert** the alarm with the specific AlarmID into the alarm list, which is sorted in order of the id. Then it prints:
    Output: "*Alarm(<alarm_id>) Inserted by Main Thread <thread-id> Into Alarm List at <insert_time>: Group(<group_id>) <time message>*"
2. **Command:** "*Change_Alarm(Alarm_ID): Group(Group_ID) Time Message*"
    a. "Alarm_ID" is a positive int.
    b. "Group" is a keyword.
    c. "Group_ID" is a positive int, to determine the group.
    d. "Time" is the amount of time till the alarm triggers.
    e. "Message" is the message displayed at the alarm's trigger.

    Change_Alarm: The main thread will insert the change alarm request with the id into a separate alarm request list, in which they are ordered by the id. Then it will print:
    Output: "*Change Alarm Request (<alarm_id>) Inserted by Main Thread<thread-id> into Alarm List at <insert_time>: Group(<group_id>) <time message>*"

1.3 - The Main Thread

The main thread has three main functions:

1. Creating the ***Alarm Monitor Thread.***
2. Determining if the format of the alarm request is correct.
    a. If the request is not consistent prompt an error
    b. Truncate to 128 characters if the message exceeds its limit.
3. Responsible for creating ***Display Alarm Thread,*** such that each display alarm thread will only display one group of alarms that have the same specific GroupID as per the requirements.

1.3.1 Main Thread Display Thread Conditions

Scenario A - Creates a new display alarm thread that will be responsible for printing the alarms of that specific group id and assigns the alarms to that newly created alarm thread. This requires two conditions:

1. There does not exist a display alarm thread, with the associated group id of the alarm; or
2. All the display alarm threads responsible for displaying the alarms with the specific group id of that alarm are already responsible for more than one alarm.
3. Prints: "*Main Thread Created New Display Alarm Thread <thread-id> For Alarm(<alarm_id> at <create_time>: Group(<group_id>) <time message>*"

Scenario B - Otherwise the main thread assigns that alarm to an existing display alarm thread associated with the same group id. It will then print:
"*Main Thread <thread-id> Assigned to Display Alarm(<alarm_id> at <assign_time>: Group(<group_id>) < time message>*"

1.4 - The Alarm Monitor Thread

1.4.1 Scenario A1 - Valid Change_Alarm Request

Condition: A matching alarm with the same Alarm_ID is found in the alarm list.
Action: The alarm monitor thread updates the Group_ID, Time, and Message of the alarm with values from the Change_Alarm request.
Notification:

- Prints: "*Alarm Monitor Thread <thread-id> Changed Alarm(<alarm_id> at <alarm_change_time>: Group(<group_id>) <time message>*".
- Removes the Change_Alarm request from the list.

1.4.2 Scenario A2 - Invalid Change_Alarm Request

Condition: No matching alarm is found.
Notification:

- Prints: "*Invalid Change Alarm Request(<alarm_id> at <invalid change alarm request detection_time>: Group(<group_id>) <time message>*".
- Removes the Change_Alarm request from the list.

1.4.3 Scenario B - Handling Expired Alarms

Condition: An alarm's expiry time is reached.
Action: The alarm monitor thread removes the expired alarm from the list.
Notification:

- Prints: "*Alarm Monitor Thread <thread-id> Removed Alarm(<alarm_id> at <remove_time>: Group(<group_id>) <time message>*".

1.5 - The Display Thread

After being created by the main thread, the display thread with the specific group id is responsible for alarms with that same id. Meaning it will print every five seconds the following:
Output: "*Alarm (<alarm_id>) Printed by Alarm Display Thread <thread-id>at <print_time>: Group(<group_id>) <time message>* ".

1.5.1 Alarm Removal Handling

If an alarm is removed from the alarm list, the responsible display alarm thread stops printing its message.
Notification: "*Display Thread <thread-id> Has Stopped Printing Message of Alarm(<alarm_id> at <stopped_print_time>: Group(<group_id>) <time message>*".

1.5.2 Group ID Change Handling

If an alarm's Group ID changes, the previous group's display thread stops printing its message.

- **Notification:** "*Display Thread <thread-id> Has Stopped Printing Message of Alarm(<alarm_id> at <old_group_change_time>: Changed Group(<group_id>) <time message>*".
- The new group's display thread starts printing the message every 5 seconds.
- **Notification:** "*Display Thread <thread-id> Has Taken Over Printing Message of Alarm(<alarm_id> at <new_group_change_time>: Changed Group(<group_id>) <time message>*".

1.5.3 Message Change Handling

If an alarm's message changes (without a Group ID change), the display thread starts printing the new message every 5 seconds.

- **Notification:** "*Display Thread <thread-id> Starts to Print Changed Message Alarm(<alarm_id> at <message_change_time>: Group(<group_id>) <time message>*".

1.5.4 No Alarms in Group Handling

If a display alarm thread finds no alarms for its group, it prints a notification and then exits.

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

1.5.4 Display Thread Conduct

- Display alarm threads must not modify any shared data structures.

1.5.5 Thread Synchronization

- Thread access to the alarm list should be treated as a "Readers-Writers" problem in "New_Alarm_Cond.c".
- Modification requires a writer process (one at a time).
- Reading allows for any number of reader processes simultaneously.

2. System Interface

2.1 - Constraints

In the program, we ensure users can input commands, choosing from two options: Start_Alarm, and Change Alarm. This is implemented with the creation using multiple threads, the main thread, display thread and alarm monitor thread. To maintain data consistency, and avoid the reader-writers problem we implement the use of semaphores to resolve this issue.

2.2 - New_Alarm_Cond

2.2.1. Struct Declarations

Struct 1 - alarm_tag: Defines a struct for storing each alarm's information.
Fields include:

- struct alarm_tag *link for linking alarms in a list.
- int group for alarm group.
- int alarm_id for unique alarm identification.
- int seconds for time in seconds.
- time_t time for time from EPOCH.
- char message[128] for alarm message.

Struct 2 - display_alarm_info: Defines a struct for display alarm thread information.
Fields include:

- pthread_t thread for the thread ID.
- int alarm_group for the alarm's group.
- int alarms_in_group for the number of alarms in the group.
- int alarm1 and int alarm2 for alarm IDs.
- int alarm1_taken_over and int alarm2_taken_over as flags.
- char alarm1_message[128] and char alarm2_message[128] for messages.
- pthread_cond_t condition for thread signaling.
- struct display_alarm_info *next for linking threads.

2.2.2. Variable Declarations

Variables:

- alarm_t *alarm_list: Linked list of alarms.
- alarm_t *changed_alarm_list: List of changed alarm requests.
- display_alarm_info_t *display_alarm_threads: List of display threads.
- pthread_mutex_t display_list_mutex: Mutex for display thread list.
- sem_t for various semaphores (display list, alarm write, read, changed alarm, monitor).
- int reading_threads: Count of threads reading the alarm list.

2.2.3. Function Declarations

Functions:

- void start_reading(): Increments thread count accessing the alarm list.
- void stop_reading(): Decrements thread count accessing the alarm list.
- void *display_alarm(void *args): Function for display alarm threads.
- void *monitor_alarms(void *args): Function for monitoring alarms.
- void process_expired_alarms(): Handles expired alarms.
- void insert_alarm_changed(alarm_t *alarm): Inserts a changed alarm.
- void check_or_create_display_thread(alarm_t *alarm, int taken_over): Manages display threads for alarms.
- void insert_alarm(alarm_t *alarm): Inserts a new alarm.

2.2.4. New_Alarm_Mulex Interface

```c
#include "errors.h"
#include <pthread.h>
#include <semaphore.h>
#include <time.h>

// Define a data structure to store information about each alarm
typedef struct alarm_tag {
  struct alarm_tag *link;
  int group;
  int alarm_id;
  int seconds;
  time_t time; /* seconds from EPOCH */
  char message[128];
} alarm_t;

// Define a structure to store information about display alarm threads
typedef struct display_alarm_info {
  pthread_t thread;
  int alarm_group;
  int alarms_in_group;
  int alarm1;
  int alarm1_taken_over;
  char alarm1_message[128];
  int alarm2;
  int alarm2_taken_over;
  char alarm2_message[128];
  struct display_alarm_info *next; // Pointer to the next thread in
the list
} display_alarm_info_t;

// List of display threads
display_alarm_info_t *display_alarm_threads;

// Mutex for display thread list, to keep track of the display thread
list
pthread_mutex_t display_list_mutex = PTHREAD_MUTEX_INITIALIZER;

sem_t display_list_semaphore;  // Semaphore for display thread list
sem_t alarm_write_semaphore;   // Semaphore to write to the alarm list
sem_t alarm_read_semaphore;    // Semaphore to read from the alarm
list
sem_t changed_alarm_semaphore; // Semaphore for changed alarm list
sem_t monitor_semaphore;       // Semaphore for monitor thread to be
signaled

alarm_t *alarm_list = NULL;
alarm_t *changed_alarm_list = NULL;

// Count of threads reading the alarm list
int reading_threads = 0;

/**
 * @brief Increments the count of threads reading the alarm list.
 *
 * Increments the count of display threads accessing the alarm list
and locks
 * the writer semaphore if this is the first display thread reading
the list.
 */
void start_reading();
/**
 * @brief Decrements the count of threads reading the alarm list.
 *
 * Decrements the count of display threads accessing the alarm list
and unlocks
 * the writer semaphore if this is the last display thread reading the
list.
 */
void stop_reading();
```

```c
/**
 * @brief The display alarm thread function that displays alarms in
its group.
 *
 * This function is responsible for displaying alarms associated
with a specific
 * group number. It checks for alarms in the specified
 * group and display them at 5 second increments.
 *
 */
void *display_alarm(void *args);

/**
 * @brief Monitors alarms for changes and expiration.
 *
 * Monitor the alarms for changes and expiration. It handles
modification
 * requests, updates, and signals the appropriate threads for
further
 * processing.
 *
 */
void *monitor_alarms(void *args);

void process_expired_alarms();

/**
 * @brief Inserts a changed alarm into the list of alarms to be
updated.
 *
 * Inserts a changed alarm into the list of alarms to be updated and
signals the
 * monitor thread to process these changes.
 *
 * @param alarm A pointer to the alarm structure that has been
modified.
 */
void insert_alarm_changed(alarm_t *alarm);

/**
 * @brief Checks for or creates a display thread for the alarm
group.
 *
 * Checks if a display thread exists for the alarm's group. If it
does, the
 * alarm is assigned to an available thread. If not, a new thread is
created for
 * the group.
 *
 * @param alarm A pointer to the alarm structure to be assigned or
displayed.
 * @param taken_over An indicator whether the alarm has been taken
over by
 * another thread.
 */
void check_or_create_display_thread(alarm_t *alarm, int taken_over);

/**
 * @brief Inserts a new alarm into the list of alarms.
 *
 * Inserts a new alarm into the list of alarms sorted by the alarm
ID. It
 * signals the monitor thread about the new alarm and checks or
creates a
 * display thread for it.
 *
 * @param alarm A pointer to the new alarm structure to be inserted.
 */
void insert_alarm(alarm_t *alarm);
```

2.3 - Alarm Monitor Thread

2.3.1. Monitor Alarms Thread Responsibilities

The monitor alarm thread monitors and updates alarms based on change requests made by the user.

Semaphore Mechanism: It uses semaphores (*monitor_semaphore* and *changed_alarm_semaphore*) to manage access to shared resources, ensuring thread-safe operations.

Concurrency Control: Uses start_reading() and stop_reading() functions to manage read access to shared alarm lists, ensuring thread safety.

Main Loop:

- ***Alarm Update*****:** Iterates through a list of changed alarms (changed_alarm_list). For each changed alarm, it finds the corresponding alarm in the main alarm list (alarm_list) and updates its properties (group, seconds, message, and time).
- ***Change Notification*****:** Prints messages to indicate the alarm changes or invalid requests.
    - “Alarm Monitor Thread %ld Has Changed Alarm(%d) at %ld: Group(%d) %d %s\n”
    - “Invalid Change Alarm Request(%d) at %ld: Group(%d) %d %s\n”
- ***List Maintenance*****:** After processing, it removes the processed alarm from changed_alarm_list.

2.3.2. Alarm Monitor Thread

```c
void *monitor_alarms(void *args) {
  // Wait for a new alarm insertion in the alarm list of changed alarm list
  sem_wait(&monitor_semaphore);

  while (1) {
    // Check for changed alarm requests and process them if any
    sem_wait(&changed_alarm_semaphore);

    while (changed_alarm_list != NULL) {
      alarm_t *current = changed_alarm_list;
      alarm_t *prev = NULL;
      start_reading(); // Lock the semaphore to access the alarm list

      while (current != NULL) {
        // Find the corresponding alarm in the alarm list
        alarm_t *alarm_to_change = NULL;
        alarm_t *alarm_iterator = alarm_list;

        while (alarm_iterator != NULL) {
          if (current->alarm_id == alarm_iterator->alarm_id) {
            alarm_to_change = alarm_iterator;
            break;
          }
          alarm_iterator = alarm_iterator->link;
        }

        if (alarm_to_change != NULL) {
          // Replace values in the corresponding alarm with Change_Alarm request
          // values
          int old_group = alarm_to_change->group;
          alarm_to_change->group = current->group;
          alarm_to_change->seconds = current->seconds;
          alarm_to_change->time = time(NULL);
          strcpy(alarm_to_change->message, current->message);

          // Print change message
          printf("Alarm Monitor Thread %ld Has Changed Alarm(%d) at %ld: "
                 "Group(%d) "
                 "%d %s\n",
                 pthread_self(), alarm_to_change->alarm_id, time(NULL),
                 alarm_to_change->group, alarm_to_change->seconds,
                 alarm_to_change->message);
          if(old_group != alarm_to_change->group){
            check_or_create_display_thread(alarm_to_change, 1);
          }
        } else {
          // Print invalid change alarm message
          printf("Invalid Change Alarm Request(%d) at %ld: Group(%d) %d %s\n",
                 current->alarm_id, time(NULL), current->group,
                 current->seconds, current->message);
        }

        // Remove the Change_Alarm request from the list
        if (prev == NULL) {
          changed_alarm_list = current->link;
          free(current);
          current = changed_alarm_list;
        } else {
          prev->link = current->link;
          free(current);
          current = prev->link;
        }
      }
      stop_reading();
    }
```

2.4 - Display Thread

2.4.1. Display Thread Responsibilities

The display thread is responsible for displaying information about alarms assigned to it, based on their group and status.

Thread and Variable Initialization: Extracts its own thread ID (*pthread_t self_id*).
Initializes local variables for storing alarm information (*local_alarm1, local_alarm2, local_alarm_group, etc.*).

Main Loop:

- The function operates in an infinite loop, sleeping for 5 seconds at the beginning of each iteration.
- Uses a semaphore (***display_list_semaphore***) to safely access a shared display alarm list.

Finding and Updating Alarms: Searches for the thread's entry in a list of display alarms (*display_alarm_threads*) to update local variables with the latest alarm data.
Locks a semaphore (*start_reading()*) to access the global alarm list (*alarm_list*).

Alarm Processing: Checks and processes two alarms (*local_alarm1 and local_alarm2*).
For each alarm, it:

- Verifies if the alarm belongs to the thread's group.
- Determines if the alarm message has changed or if the thread has taken over the alarm.
- Prints appropriate messages based on the alarm's status (*changed, taken over, stopped printing, etc.*).
- Updates local variables and thread information as needed.

Exiting Conditions: If both alarms (*local_alarm1 and local_alarm2*) are reassigned or no longer relevant, the thread:

- Prints a message indicating it's exiting.
- Removes its entry from the display alarm list.
- Frees associated memory and exits using pthread_exit(*NULL*).

Semaphore Management:

- Unlocks (*stop_reading()*) the semaphore after accessing the alarm list.
- Updates the thread's alarm information in the display alarm list.
- Releases the **display_list_semaphore** at the end of each iteration.

2.4.2. Display Thread

```
void *display_alarm(void *args) {
  // Extract the thread ID and alarm group for this display thread
  pthread_t self_id = pthread_self();

  // Local variables to store information
  int local_alarm1 = -1, local_alarm2 = -1;

  int local_alarm_group = -1;
  int local_alarm1_taken_over = 0, local_alarm2_taken_over = 0;
  char local_alarm1_message[128];
  char local_alarm2_message[128];

  while (1) {
    // Sleep for 5 seconds first
    sleep(5);

    // Lock the display list semaphore to access the display alarm
list
    sem_wait(&display_list_semaphore);

    // Search for this thread's entry in the display alarm list
    display_alarm_info_t *current_thread = display_alarm_threads;
    display_alarm_info_t *prev_thread = NULL;

    while (current_thread != NULL) {
      if (pthread_equal(current_thread->thread, self_id)) {
        // Found the current thread's entry in the list, update local
variables
        local_alarm_group = current_thread->alarm_group;
        local_alarm1 = current_thread->alarm1;
        local_alarm2 = current_thread->alarm2;
        local_alarm1_taken_over = current_thread->alarm1_taken_over;
        local_alarm2_taken_over = current_thread->alarm2_taken_over;
        strcpy(local_alarm1_message, current_thread->alarm1_message);
        strcpy(local_alarm2_message, current_thread->alarm2_message);
        break;
      }
      prev_thread = current_thread;
      current_thread = current_thread->next;
    }

    int old_local_alarm1 = local_alarm1;
    int old_local_alarm2 = local_alarm2;

    // Lock the semaphore to access the alarm list
    start_reading();

    // Check if local_alarm1 is not -1
    if (local_alarm1 != -1) {
      alarm_t *current_alarm = alarm_list;
      int alarm1_found = 0;
      while (current_alarm != NULL) {
        if (current_alarm->alarm_id == local_alarm1) {
          if (current_alarm->group == local_alarm_group) {
            if (local_alarm1_taken_over) {
              // Taken over alarm
              // Print the alarm message not taken oven
              printf("Display Thread %lu Has Taken Over Printing
Message of "
                     "Alarm(%d) at %ld: Changed Group(%d) %d %s\n",
                     self_id, current_alarm->alarm_id, time(NULL),
                     local_alarm_group, current_alarm->seconds,
                     current_alarm->message);
              local_alarm1_taken_over = 0;
              alarm1_found = 1;
            }
else {
              if (strcmp(local_alarm1_message,
current_alarm->message) != 0) {
                // Message changed
                printf("Display Thread %lu Starts to Print
Changed Message of "
                       "Alarm(%d) at %ld: Group(%d) %d
%s\n",
                       self_id, current_alarm->alarm_id,
time(NULL),
                       local_alarm_group,
current_alarm->seconds,
                       current_alarm->message);
                local_alarm1_taken_over = 0;
                alarm1_found = 1;
                // Copy current_alarm->message to
local_alarm1_message
                strcpy(local_alarm1_message,
current_alarm->message);

              } else {
                // Print the alarm message not taken oven,
same message
                printf("Alarm (%d) Printed by Alarm
Display Thread %lu at %ld: "
                       "Group(%d) %d %s\n",
                       local_alarm1, self_id, time(NULL),
local_alarm_group,
                       current_alarm->seconds,
current_alarm->message);
              }
            }
            alarm1_found = 1;
            break;
          } else {
            // Alarm found but group has changed,
            printf("Display Thread %lu Has Stopped
Printing Message of "
                   "Alarm(%d) at %ld: Changed Group(%d)
%s\n",
                   self_id, local_alarm1, time(NULL),
local_alarm_group,
                   local_alarm1_message);
            // Update local variables and thread
information
            current_thread->alarms_in_group--;
            alarm1_found = 1;
            local_alarm1 = -1;
            local_alarm1_taken_over = -1;
            break;
          }
        }
        current_alarm = current_alarm->link;
      }
// Alarm removed from list
      if (!alarm1_found) {
        // alarm1 is no longer in the alarm
        printf("Display Thread %lu Has Stopped Printing Message of 
Alarm(%d) "
               "at %ld: Group(%d) %s\n",
               self_id, local_alarm1, time(NULL), local_alarm_group,
               local_alarm1_message);
        // Update local variables and thread information
        current_thread->alarms_in_group--;
        local_alarm1 = -1;
        local_alarm1_taken_over = -1;
      }
    }

    // Similarly, perform the same steps for local_alarm2
    // Check if local_alarm2 is not -1
    if (local_alarm2 != -1) {
      alarm_t *current_alarm = alarm_list;
      int alarm2_found = 0;
      while (current_alarm != NULL) {
        if (current_alarm->alarm_id == local_alarm2) {
          if (current_alarm->group == local_alarm_group) {
            if (local_alarm2_taken_over) {
              // Taken over alarm
              // Print the alarm message not taken over
              printf("Display Thread %lu Has Taken Over Printing 
Message of "
                     "Alarm(%d) at %ld: Changed Group(%d) %d %s\n",
                     self_id, current_alarm->alarm_id, time(NULL),
                     local_alarm_group, current_alarm->seconds,
                     current_alarm->message);
              local_alarm2_taken_over = 0;
              alarm2_found = 1;
            } else {
              if (strcmp(local_alarm2_message, current_alarm->message) 
!= 0) {
                // Message changed
                printf("Display Thread %lu Starts to Print Changed 
Message of "
                       "Alarm(%d) at %ld: Group(%d) %d %s\n",
                       self_id, current_alarm->alarm_id, time(NULL),
                       local_alarm_group, current_alarm->seconds,
                       current_alarm->message);
                local_alarm2_taken_over = 0;
                alarm2_found = 1;
                // Copy current_alarm->message to local_alarm2_message
                strcpy(local_alarm2_message, current_alarm->message);
              } else {
                // Print the alarm message not taken over, same 
message
                printf("Alarm (%d) Printed by Alarm Display Thread %lu 
at %ld: "
                       "Group(%d) %d %s\n",
                       local_alarm2, self_id, time(NULL), 
local_alarm_group,
                       current_alarm->seconds, 
current_alarm->message);
              }
            }
            alarm2_found = 1;
            break;
          }
else {
            // Alarm found but group has changed,
            printf("Display Thread %lu Has Stopped Printing Message 
of "
                   "Alarm(%d) at %ld: Changed Group(%d) %s\n",
                   self_id, local_alarm2, time(NULL), 
local_alarm_group,
                   local_alarm2_message);
            // Update local variables and thread information
            current_thread->alarms_in_group--;
            alarm2_found = 1;
            local_alarm2 = -1;
            local_alarm2_taken_over = -1;
            break;
          }
        }
        current_alarm = current_alarm->link;
      }
      // Alarm removed from list
      if (!alarm2_found) {
        // alarm2 is no longer in the alarm
        printf("Display Thread %lu Has Stopped Printing Message of 
Alarm(%d) "
               "at %ld: Group(%d) %s\n",
               self_id, local_alarm2, time(NULL), local_alarm_group,
               local_alarm2_message);
        // Update local variables and thread information
        current_thread->alarms_in_group--;
        local_alarm2 = -1;
        local_alarm2_taken_over = -1;
      }
    }
    // Unlock the semaphore to release the alarm list
    stop_reading();

    // Check if both alarms were reassigned
    if (local_alarm1 == -1 && local_alarm2 == -1) {
      printf(
          "No More Alarms in Group(%d): Display Thread %ld exiting 
at %lu.\n",
          local_alarm_group, self_id, time(NULL));
      // Update pointers to remove the node
      if (prev_thread != NULL) {
        prev_thread->next = current_thread->next;
      } else {
        // If the current thread is the head of the list
        display_alarm_threads = current_thread->next;
      }
      // Free memory from the current thread
      free(current_thread);
      // Release the display list semaphore after accessing the list
      sem_post(&display_list_semaphore);
      pthread_exit(NULL); // Terminate the thread
    }

    // Update the content of the thread
    current_thread->alarm1 = local_alarm1;
    current_thread->alarm2 = local_alarm2;
    current_thread->alarm1_taken_over = local_alarm1_taken_over;
    current_thread->alarm2_taken_over = local_alarm2_taken_over;
    strcpy(current_thread->alarm1_message, local_alarm1_message);
    strcpy(current_thread->alarm2_message, local_alarm2_message);
    // Release the display list semaphore after accessing the list
    sem_post(&display_list_semaphore);
  }

  return NULL;
}
```

3. Challenges

This assignment includes managing concurrent access to shared data structures (e.g., alarm lists) by multiple threads, ensuring synchronization and avoiding race conditions. Additionally, handling dynamic thread creation and assignment to display alarm threads requires careful management of thread lifecycles and resources. Ensuring proper error handling and debugging for scenarios like invalid input or thread-related issues is essential for the successful implementation of this multithreaded alarm management system.

Handling many different data structures proved to be more challenging than expected. Ensuring the program can handle the input since the last assignment was dealing with one thread as compared to now.

3.1 - Issues Arisen

Starvation: Starvation occurs when the two or more threads that are allocated to the CPU take excessive time in execution, causing other threads not able to be processed.

- If the main thread has to wait for a long time to get the lock from the display thread, if the display thread is doing constant reading.

Synchronization: We needed a mechanism that allowed us to read simultaneously while ensuring exclusive access for writers of the main display threads or alarm monitor thread during the operations.

- **Concurrency management:** Utilizing a reader-writer lock mechanism to enable simultaneous reading by multiple display threads.
- Using the POSIX functions ensures data consistency and prevents interference among the other threads facilitating the multithreaded alarm management system.

Readers-Writers-Problem: This is a problem that is addressed when a shared resource (alarm) can be accessed by multiple readers, and writers. The issue that would occur if not addressed would be the following:

- **Inconsistent Data Access:** Without proper synchronization, multiple display threads (readers) could read alarm data while it is being modified by a writer thread (either the main thread creating/modifying alarms or the alarm monitor thread managing expirations). This could lead to display threads reading inconsistent or partial data, leading to incorrect alarm information being shown.
- **Race Conditions:** If a writer thread is updating alarm information (e.g., changing an alarm time or message) while a reader thread is accessing the same alarm, it could lead to a race condition. This means the final state of the alarm data could depend on the sequence of operations performed by the reader and writer threads, leading to unpredictable results.
- **Starvation of Writers:** In a scenario where reader threads are frequently accessing alarm data, writer threads might experience starvation — a situation where they are continually waiting for access to the shared resource. This can occur if new alarms are constantly being read and displayed, but updates to alarms or new alarm insertions are perpetually delayed because the readers keep the resource occupied.

- **Priority Inversion:** This happens when higher priority tasks (like updating or removing an expired alarm) are delayed because lower priority tasks (like reading alarms for display) hold the necessary resources. Without a proper solution, this could lead to inefficient handling of alarms, especially in time-sensitive situations.

4. System Design

*(**Note**: Other than the alarm_cond.c program sourced from "Programming with POSIX Threads" by David R. Butenhof the code we provided was written by our group and the explanations/examples provided below are our group's own work.)*

4.1 - Synchronization

Our core synchronization method for our system uses semaphores which allows us to overcome the busy waiting problem of using mutexes when waiting on a locked mutex and instead when threads are waiting on a locked semaphore they enter a waiting queue and are not talking up processing power. Additionally, semaphores provide the means for signaling between threads, which are employed for communication between the main thread and the monitor thread.

4.2 - Readers-Writers Model

4.2.1 Shared Resources

The shared resources in this context are the alarm list, the changed alarm requests list, and the display thread list.

4.2.2 Thread Responsibilities

Our multithreaded system is divided into:

- **Main thread:** responsible for adding alarms into the alarm list, change alarm requests into the change alarm list and creating or assigning display threads with the appropriate alarms.
- **Monitor thread:** responsible for applying the changes requests in the change alarm list and to removing expiring alarms from the alarm list.
- **Display threads:** responsible for printing the message and detecting changes for the alarms they are responsible for.

4.3 Monitor Thread

In our system implementation the monitor thread is initially sleeping and waits on a change alarm request to process or an alarm in the list to be inserted which it would need to remove when the alarm eventually expires. The monitor thread when signaled will attempt to apply all change requests immediately when received if they are valid. Additionally, the monitor thread will perform a timed wait on the monitor semaphore for a time period until the nearest expiring alarm or unless the monitor semaphore is signaled for a change request. Using this approach greatly improves the efficiency and number of threads running as the monitor thread would only be running when it requires to process change requests or an alarm expires and needs to be removed otherwise the monitor thread is kept on the wait queue for the monitor semaphore.

4.4 - Display Threads

The display threads keep track of their entry in the display thread list which store information on the group number, alarm, alarm messages and the alarm taken over flag for each of the two alarms they are responsible for. If an alarm is set to -1 in the display thread struct this indicates an open alarm slot for that display thread. The display thread checks the display thread list (locks and unlocks the the display list semaphore) and updates its local data, then it checks for the alarms its responsible for and detects any changes in the message and group number for that alarm and prints the corresponding alarm message every 5 seconds until the alarm expires or its group changes due to a change request. If while checking display thread finds it is no longer responsible for any alarms it removes its entry from the display alarm list and frees its memory. The display threads are set to sleep for periods of 5 seconds between checks ensuring that they have unlocked the display alarm list semaphore prior and that they signaled that they stopped reading the alarm list beforehand. This sleep period allows for less contention between the display threads reading the alarm list and main and monitor threads writing or changing the alarm list. One of the setbacks to this approach is a delay for the display thread to check that's no longer responsible for any alarms however, since the display thread is sleeping it would contend on a shared resource with another thread and there would never be a cases where are an alarm is assigned to two display threads so it's hidden from a user's perspective as the message is still printed every 5 seconds and alarms are corr at the correct time.

4.5 - Readers and Writers

Display threads act as readers when they periodically check for the display thread list for any new assignments and the alarm list to check if the alarms they are assigned still exist, have been changed or need to be printed. However, they act as readers when they update the display thread list when an alarm they find an alarm they are no longer responsible for exists or that they are longer responsible for any alarms where they would remove their entry from the list and exit. The monitor thread acts as a reader and a writer as it reads change alarm requests, reads the alarm list to find the appropriate entry, writes to the alarm list the changes and removes the change alarm request from the change alarm requests list. The main thread serves mainly as a writer, updating the list when new alarms are created, change requests are created, and when display threads are assigned. Based on their responsibilities and the shared resources they need the change alarm list and the display thread list only need a semaphore to access these resources as at most one reader/writer is needed for them at a time.

However, based on this design the only resource that would need a synchronization method for multiple readers and writers is the alarm list as there can be multiple threads trying to read alarms but would not change any of the information of the alarms themselves and similarly based on our design the monitor thread is set to do a timed wait (to improve processing efficiency) to the time of the next expiring alarm and to check for the nearest expiring alarm it would only need to read the list and not make any changes.

```
void start_reading() {
  // Increment the count of display
threads
  sem_wait(&alarm_read_semaphore);
  reading_threads++;
  // If this is the first display thread,
lock the alarm (writer) semaphore
  if (reading_threads == 1) {
    sem_wait(&alarm_write_semaphore);
  }
  sem_post(&alarm_read_semaphore);
}
```

```
void stop_reading() {
  sem_wait(&alarm_read_semaphore);
  reading_threads--;
  // Last display thread
  if (reading_threads == 0) {
    sem_post(&alarm_write_semaphore);
  }
  sem_post(&alarm_read_semaphore);
}
```

As seen above our solution to this problem is for our system to use a reader semaphore keep track of a counter for the number of readers are currently reading (reading_threads int variable) and if there are more than one reader the write semaphore (**alarm_write_semaphore**) is locked to ensure that the shared alarm list remains consistent in memory when read and is not changed midway. This also would support the case where if a change was currently being applied the reader thread would have to wait for the writer to be unlocked first before having a chance to read. Similarly, when the last reader is done reading the list it unlocks the writer semaphore allowing for changes to be made to the alarm list since no other reading is currently reading from it.

In our implementation each of the display threads first calls start_reading() before reading from the alarm list and calls stop_reading() when done reading which takes care of incrementing and decrementing the number of threads currently reading and locking and unlocking the writer semaphore appropriately. The alarm monitor thread and main thread use the writer semaphore to write to the alarm list when a new alarm gets inserted or an alarm is changed respectively. The monitor thread also calls start_reading() and stop_reading() when it needs to find the nearest expiring alarm as it only needs to read the list at that point and not make any changes. The main thread and alarm monitor thread when inserting alarms or making changes to alarms use the **alarm_write_semaphore** when trying to change information in the alarm list. Effectively, this system allows only one writer to write the list and allows multiple threads to read from the alarm list at the same time and by using the sleep time it gives priority for the main and monitor writer threads to perform their actions immediately or as fast as possible .

5. Test Cases

5.1 - Scenarios

In output.txt, the scenarios were tested, and the exact outputs were placed there.

Scenario 1: Testing Start_Alarm command.
Inputs:

- **Time 0:** "*Start_Alarm(1): Group(1) 10 Test_Message*"

Expected Outputs:

- **Time 0:** *Alarm(1) Inserted by Main Thread <ThreadID> Into Alarm List at <TimeFromEpoch>: Group(1) 10 Test_Message*
- **Time 0:** *Main Thread Created New Display Alarm Thread 0 For Alarm(1) at <TimeFromEpoch>: Group(1) 10 Test_Message*
- **Time 5:** *Alarm (1) Printed by Alarm Display Thread <ThreadID> at <TimeFromEpoch>: Group(1) 10 Test_Message*
- **Time 5:** *Alarm Monitor Thread <ThreadID> Has Removed Alarm(1) at <TimeFromEpoch>: Group(1) 10 Test_Message*
- **Time 10:** *Display Thread <ThreadID> Has Stopped Printing Message of Alarm(1) at 1701653025: Group(1) Test_Message*
- **Time 10:** *No More Alarms in Group(1): Display Thread <ThreadID> exiting at <TimeFromEpoch>.*

Actual Output:

```
Alarm> Start_Alarm(1): Group(1) 10 Test_Message

Alarm(1) Inserted by Main Thread 140170938833792 Into Alarm List at 1701653014: Group(1) 10
Test_Message

Main Thread Created New Display Alarm Thread 140170930435648 For Alarm(1) at 1701653014:
Group(1) 10 Test_Message

Alarm> Alarm (1) Printed by Alarm Display Thread 140170930435648 at 1701653019: Group(1) 10
Test_Message

Alarm Monitor Thread 140170938828352 Has Removed Alarm(1) at 1701653024: Group(1) 10
Test_Message

Display Thread 140170930435648 Has Stopped Printing Message of Alarm(1) at 1701653025:
Group(1) Test_Message
No More Alarms in Group(1): Display Thread 140170930435648 exiting at 1701653025.
```

Scenario 2: Testing Start_Alarm command with negative id.
Inputs:

- **Time 0:** "*Start_Alarm(-1): Group(1) 20 Test_Message*"

Expected Outputs:

- **Time 0:** *"Alarm ID must be greater than or equal to 0 0"*

Actual Output:

```
Alarm>
Start_Alarm(-1): Group(1) 50 Testing
Alarm ID must be greater than or equal to 0 0
```

Scenario 3: Testing Change_Alarm command.

Inputs:

- **Time 0:** "*Start_Alarm(1): Group(1) 10 Test_Message*"
- **Time 5:** "*Change_Alarm(1): Group(1) 10 Replaced_Message*"

Expected Outputs:

- **Time 0:** *Alarm(1) Inserted by Main Thread <ThreadID> Into Alarm List at <TimeFromEpoch>: Group(1) 10 Test_Message*
- **Time 0:** *Main Thread Created New Display Alarm Thread 0 For Alarm(1) at <TimeFromEpoch>: Group(1) 10 Test_Message*
- **Time 5:** *Alarm (1) Printed by Alarm Display Thread <ThreadID> at <TimeFromEpoch>: Group(1) 10 Test_Message*
- **Time 5:** *Alarm Monitor Thread <ThreadID> Has Removed Alarm(1) at <TimeFromEpoch>: Group(1) 10 Test_Message*
- **Time 5:** *Alarm (1) Printed by Alarm Display Thread <ThreadID> at <TimeFromEpoch>: Group(1) 10 Test_Message*
- **Time 6:** *Change Alarm Request(1) Inserted by Main Thread <ThreadID> into Alarm List at <TimeFromEpoch>: Group(1) 10 Replaced_Message*
- **Time 6:** *Alarm>Alarm Monitor Thread <ThreadID> Has Changed Alarm(1) at <TimeFromEpoch>: Group(1) 10 Replaced_Message*
- **Time 10:** *Display Thread <ThreadID> Starts to Print Changed Message of Alarm(1) at <TimeFromEpoch>: Group(1) 10 Replaced_Message*
- **Time 15:** *Alarm (1) Printed by Alarm Display Thread <ThreadID> at <TimeFromEpoch>: Group(1) 10 Replaced_Message*
- **Time 16:** *Alarm Monitor Thread <ThreadID> Has Removed Alarm(1) at <TimeFromEpoch>: Group(1) 10 Replaced_Message*
- **Time 20:** *Display Thread <ThreadID> Has Stopped Printing Message of Alarm(1) at <TimeFromEpoch>: Group(1) Replaced_Message*
- **Time 20:** *No More Alarms in Group(1): Display Thread <ThreadID> exiting at <TimeFromEpoch>.*

Actual Output:

```
Alarm>Start_Alarm(1): Group(1) 10 Test_Message

Alarm(1) Inserted by Main Thread 139696601275264 Into Alarm List at 1701653762: Group(1) 10
Test_Message

Main Thread Created New Display Alarm Thread 139696592877120 For Alarm(1) at 1701653762:
Group(1) 10 Test_Message
```

```
Alarm>Change_Alarm(1): Group(1) 10 Replaced_MessageAlarm (1) Printed by Alarm Display Thread
139696592877120 at 1701653767: Group(1) 10 Test_Message

Change Alarm Request(1) Inserted by Main Thread 139696601275264 into Alarm List at 1701653768:
Group(1) 10 Replaced_Message

Alarm>Alarm Monitor Thread 139696601269824 Has Changed Alarm(1) at 1701653768: Group(1) 10
Replaced_Message

Display Thread 139696592877120 Starts to Print Changed Message of Alarm(1) at 1701653772:
Group(1) 10 Replaced_Message
Alarm (1) Printed by Alarm Display Thread 139696592877120 at 1701653777: Group(1) 10
Replaced_Message

Alarm Monitor Thread 139696601269824 Has Removed Alarm(1) at 1701653778: Group(1) 10
Replaced_Message

Display Thread 139696592877120 Has Stopped Printing Message of Alarm(1) at 1701653782:
Group(1) Replaced_Message

No More Alarms in Group(1): Display Thread 139696592877120 exiting at 1701653782.
```

Scenario 4: Testing Start_Alarm command with negative group id.
Inputs:

- **Time 0:** "*Start_Alarm(1): Group(-1) 20 Test_Message*"

Expected Outputs:

- **Time 0: "***Group ID must be greater than or equal to 0 0"*

Actual Output:

```
Alarm>
Start_Alarm(-1): Group(1) 50 Testing
Alarm ID must be greater than or equal to 0 0
```

Scenario 5: Inserting an alarm with an already existing ID.
Inputs:

- **Time 0: "***Start_Alarm(1): Group(1) 50 Testing"*
- **Time 1: "***Start_Alarm(1): Group(1) 50 Testing"*

Expected Outputs:

- **Time 0:** *"Alarm(1) Inserted by Main Thread <ThreadID> Into Alarm List at <TimeFromEpoch>: Group(1) 50 Testing"*
- **Time 0:** *"Main Thread Created New Display Alarm Thread <ThreadID> For Alarm(1) at <TimeFromEpoch>: Group(1) 50 Testing"*
- **Time 1:** *"An alarm with ID 1 already exists."*

Actual Output:

```
Alarm>Start_Alarm(1): Group(1) 50 Testing
```

```
Alarm(1) Inserted by Main Thread 139899185851264 Into Alarm List at 1701652819: Group(1) 50
Testing

Main Thread Created New Display Alarm Thread 139899177453120 For Alarm(1) at 1701652819:
Group(1) 50 Testing

Alarm>Start_Alarm(1): Group(1) 50 Testing

An alarm with ID 1 already exists.
```

Scenario 6: Inserting two alarms into the same group.
Inputs:

- **Time 0: "***Start_Alarm(1): Group(1) 10 Testing"*
- **Time 1: "***Start_Alarm(2): Group(1) 10 Testing"*

Expected Outputs:

- **Time 0:** *"Alarm(1) Inserted by Main Thread <ThreadID> Into Alarm List at <TimeFromEpoch>: Group(1) 10 Testing"*
- **Time 0:** *"Main Thread Created New Display Alarm Thread <ThreadID> For Alarm(1) at <TimeFromEpoch>: Group(1) 10 Testing"*
- **Time 1:** *"Alarm(1) Inserted by Main Thread <ThreadID> Into Alarm List at <TimeFromEpoch>: Group(2) 10 Testing"*
- **Time 1:** *"Main Thread Created New Display Alarm Thread <ThreadID> For Alarm(1) at <TimeFromEpoch>: Group(2) 10 Testing"*
- **Time 5:** *"Alarm (1) Printed by Alarm Display Thread <ThreadID> at <TimeFromEpoch>: Group(1) 10 Testing"*
- **Time 5:** *"Alarm (2) Printed by Alarm Display Thread <ThreadID> at <TimeFromEpoch>: Group(1) 10 Testing*"
- **Time 10:** *"Alarm Monitor Thread <ThreadID> Has Removed Alarm(1) at <TimeFromEpoch>: Group(1) 10 Testing"*
- **Time 10:** *"Display Thread <ThreadID> Has Stopped Printing Message of Alarm(1) at 1701653612: Group(1) Testing"*
- **Time 10:** *"Alarm (2) Printed by Alarm Display Thread <ThreadID> at <TimeFromEpoch>: Group(1) 10 Testing"*
- **Time 11:** *"Alarm Monitor Thread <ThreadID> Has Removed Alarm(2) at <TimeFromEpoch>: Group(1) 10 Testing"*
- **Time 11:** *"Display Thread <ThreadID> Has Stopped Printing Message of Alarm(2) at <TimeFromEpoch>: Group(1) Testing"*
- **Time 11:** *"No More Alarms in Group(1): Display Thread <ThreadID> exiting at <TimeFromEpoch>."*

Actual Output:

```
Alarm> Start_Alarm(1): Group(1) 10 Testing

Alarm(1) Inserted by Main Thread 139853826231168 Into Alarm List at 1701653602: Group(1) 10
Testing
```

```
Main Thread Created New Display Alarm Thread 139853817833024 For Alarm(1) at 1701653602:
Group(1) 10 Testing

Alarm> Start_Alarm(2): Group(1) 10 Testing

Alarm(2) Inserted by Main Thread 139853826231168 Into Alarm List at 1701653603: Group(1) 10
Testing

Main Thread 139853826231168 Assigned to Display Alarm Thread 139853817833024 at 1701653603:
Group(1) 10 Testing

Alarm>
Alarm (1) Printed by Alarm Display Thread 139853817833024 at 1701653607: Group(1) 10 Testing

Alarm (2) Printed by Alarm Display Thread 139853817833024 at 1701653607: Group(1) 10 Testing

Alarm Monitor Thread 139853826225728 Has Removed Alarm(1) at 1701653612: Group(1) 10
Testing

Display Thread 139853817833024 Has Stopped Printing Message of Alarm(1) at 1701653612:
Group(1) Testing

Alarm (2) Printed by Alarm Display Thread 139853817833024 at 1701653612: Group(1) 10 Testing

Alarm Monitor Thread 139853826225728 Has Removed Alarm(2) at 1701653613: Group(1) 10
Testing

Display Thread 139853817833024 Has Stopped Printing Message of Alarm(2) at 1701653617:
Group(1) Testing

No More Alarms in Group(1): Display Thread 139853817833024 exiting at 1701653617.
```

Scenario 7: Inserting two alarms with the same id into different groups.
Inputs:

- **Time 0: "***Start_Alarm(1): Group(1) 10 Testing"*
- **Time 1: "***Start_Alarm(1): Group(2) 10 Testing"*

Expected Outputs:

- **Time 0:** *"Alarm(1) Inserted by Main Thread <ThreadID> Into Alarm List at <TimeFromEpoch>: Group(1) 10 Testing"*
- **Time 0:** *"Main Thread Created New Display Alarm Thread <ThreadID> For Alarm(1) at <TimeFromEpoch>: Group(1) 10 Testing"*
- **Time 1:** *"An alarm with ID 1 already exists."*

Actual Output:

```
Alarm>Start_Alarm(1): Group(1) 10 Testing
```

```
Alarm(1) Inserted by Main Thread 140344571972480 Into Alarm List at 1701654431: Group(1) 10
Testing

Main Thread Created New Display Alarm Thread 140344563574336 For Alarm(1) at 1701654431:
Group(1) 10 Testing

Alarm>Start_Alarm(1): Group(2) 10 Testing

An alarm with ID 1 already exists.
```

Scenario 8: Inserting an alarm with the negative time.
Inputs:

- **Time 0: "***Start_Alarm(1): Group(1) -1 Testing"*

Expected Outputs:

- **Time 0:** *"Alarm time must be greater than 0"*

Actual Output:

```
Alarm>Start_Alarm(1): Group(1) -1 Testing

Alarm time must be greater than 0
```

Scenario 9: Inserting three alarms into 1 group.
Inputs:

- **Time 0: "***Start_Alarm(1): Group(1) 10 Testing"*
- **Time 1: "***Start_Alarm(2): Group(1) 10 Testing"*
- **Time 7: "***Start_Alarm(3): Group(1) 10 Testing"*

Expected Outputs:

- **Time 0:** *"Alarm(1) Inserted by Main Thread <ThreadID> Into Alarm List at <TimeFromEpoch>: Group(1) 10 Testing"*
- **Time 0:** *"Main Thread Created New Display Alarm Thread <ThreadID> For Alarm(1) at <TimeFromEpoch>: Group(1) 10 Testing"*
- **Time 1:** *"Alarm(2) Inserted by Main Thread <ThreadID> Into Alarm List at <TimeFromEpoch>: Group(1) 10 Testing"*
- **Time 1:** *"Main Thread <ThreadID> Assigned to Display Alarm Thread <ThreadID> at <TimeFromEpoch>: Group(1) 10 Testing"*
- **Time 5:** *"Alarm (1) Printed by Alarm Display Thread <ThreadID> at <TimeFromEpoch>: Group(1) 10 Testing"*
- **Time 5:** *"Alarm (2) Printed by Alarm Display Thread <ThreadID> at <TimeFromEpoch>: Group(1) 10 Testing*"
- **Time 7:** *"Alarm(3) Inserted by Main Thread <ThreadID> Into Alarm List at <TimeFromEpoch>: Group(1) 10 Testing"*

- **Time 7:** *"Main Thread Created New Display Alarm Thread <ThreadID> For Alarm(3) at <TimeFromEpoch>: Group(1) 10 Testing"*
- **Time 10:** *"Alarm Monitor Thread <ThreadID> Has Removed Alarm(1) at <TimeFromEpoch>: Group(1) 10 Testing"*
- **Time 10:** *"Display Thread <ThreadID> Has Stopped Printing Message of Alarm(1) at <TimeFromEpoch>: Group(1) Testing"*
- **Time 10:** *"Alarm (2) Printed by Alarm Display Thread <ThreadID> at <TimeFromEpoch>: Group(1) 10 Testing"*
- **Time 11:** *"Alarm Monitor Thread <ThreadID> Has Removed Alarm(2) at <TimeFromEpoch>: Group(1) 10 Testing"*
- **Time 11:** *"Display Thread <ThreadID> Has Stopped Printing Message of Alarm(2) at <TimeFromEpoch>: Group(1) Testing"*
- **Time 11:** *"No More Alarms in Group(1): Display Thread <ThreadID> exiting at <TimeFromEpoch>."*
- **Time 12:** *"Alarm (3) Printed by Alarm Display Thread <ThreadID> at <TimeFromEpoch>: Group(1) 10 Testing"*
- **Time 15:** *"Display Thread <ThreadID> Has Stopped Printing Message of Alarm(2) at <TimeFromEpoch>: Group(1) Testing"*
- **Time 15:** *"No More Alarms in Group(1): Display Thread <ThreadID> exiting at <TimeFromEpoch>."*
- **Time 17:***"Alarm Monitor Thread <ThreadID> Has Removed Alarm(3) at <TimeFromEpoch>: Group(1) 10 Testing"*
- **Time 17:***"Display Thread <ThreadID> Has Stopped Printing Message of Alarm(3) at <TimeFromEpoch>: Group(1) Testing"*
- **Time 17:***"No More Alarms in Group(1): Display Thread <ThreadID> exiting at <TimeFromEpoch>."*

Actual Output:

```
Start_Alarm(1): Group(1) 10 Testing

Alarm(1) Inserted by Main Thread 139894879812480 Into Alarm List at 1701797641: Group(1) 10
Testing

Main Thread Created New Display Alarm Thread 139894871414336 For Alarm(1) at 1701797641:
Group(1) 10 Testing

Alarm> Start_Alarm(2): Group(1) 10 Testing

Alarm(2) Inserted by Main Thread 139894879812480 Into Alarm List at 1701797642: Group(1) 10
Testing

Main Thread 139894879812480 Assigned to Display Alarm Thread 139894871414336 at 1701797642:
Group(1) 10 Testing

Alarm (1) Printed by Alarm Display Thread 139894871414336 at 1701797646: Group(1) 10 Testing
Alarm (2) Printed by Alarm Display Thread 139894871414336 at 1701797646: Group(1) 10 Testing

Alarm> Start_Alarm(3): Group(1) 10 Testing

Alarm(3) Inserted by Main Thread 139894879812480 Into Alarm List at 1701797648: Group(1) 10
Testing

Main Thread Created New Display Alarm Thread 139894863021632 For Alarm(3) at 1701797648:
Group(1) 10 Testing

Alarm Monitor Thread 139894879807040 Has Removed Alarm(1) at 1701797651: Group(1) 10
Testing

Display Thread 139894871414336 Has Stopped Printing Message of Alarm(1) at 1701797651:
Group(1) Testing

Alarm (2) Printed by Alarm Display Thread 139894871414336 at 1701797651: Group(1) 10 Testing

Alarm Monitor Thread 139894879807040 Has Removed Alarm(2) at 1701797652: Group(1) 10
Testing

Alarm (3) Printed by Alarm Display Thread 139894863021632 at 1701797653: Group(1) 10 Testing

Display Thread 139894871414336 Has Stopped Printing Message of Alarm(2) at 1701797656:
Group(1) Testing

No More Alarms in Group(1): Display Thread 139894871414336 exiting at 1701797656.

Alarm Monitor Thread 139894879807040 Has Removed Alarm(3) at 1701797658: Group(1) 10
Testing

Display Thread 139894863021632 Has Stopped Printing Message of Alarm(3) at 1701797658:
Group(1) Testing

No More Alarms in Group(1): Display Thread 139894863021632 exiting at 1701797658.
```

Scenario 10: Invalid Chnage_Alarm call.
Inputs:
- **Time 0: "***Change_Alarm(3): Group(1) 10 Testing"*

Expected Outputs:
- **Time 0:** *"Change Alarm Request(3) Inserted by Main Thread <ThreadID> Into Alarm List at <TimeFromEpoch>: Group(1) 10 Testing"*
- **Time 0:** *" Invalid Change Alarm Request(3) at <TimeFromEpoch>: Group(1) 10 Testing"*

Actual Output:

```
Alarm> Change_Alarm(3): Group(1) 10 Testing
```

```
Change Alarm Request(3) Inserted by Main Thread 139896472923008 into Alarm List at 1701816004:

Group(1) 10 Testing

Alarm> Invalid Change Alarm Request(3) at 1701816004: Group(1) 10 Testing
```

Scenario 11: Changing alarm at the same time it is removed.
Input:

- **Time 0:** *"Start_Alarm(3): Group(1) 10 Testing"*
- **Time 10:** *"Change_Alarm(3): Group(1) 10 new Alarm"*

Expected Output:

- **Time 0:** *"Alarm(3) Inserted by Main Thread <ThreadID> Into Alarm List at <TimeFromEpoch>: Group(1) 10 Testing"*
- **Time 0:** *"Main Thread Created New Display Alarm Thread <ThreadID> For Alarm(3) at <TimeFromEpoch>: Group(1) 10 Testing"*
- **Time 5:** *"Alarm (3) Printed by Alarm Display Thread <ThreadID> at <TimeFromEpoch>: Group(1) 10 Testing"*
- **Time 10:** *"Alarm Monitor Thread <ThreadID> Has Removed Alarm(3) at <TimeFromEpoch>: Group(1) 10 Testing"*
- **Time 10:** *"Change Alarm Request(3) Inserted by Main Thread <ThreadID> into Alarm List at <TimeFromEpoch>: Group(1) 10 new Alarm"*
- **Time 10:** *"Invalid Change Alarm Request(3) at <TimeFromEpoch>: Group(1) 10 new Alarm"*
- **Time 10:** *"Display Thread 1<ThreadID> Has Stopped Printing Message of Alarm(3) at <TimeFromEpoch>: Group(1) Testing"*
- **Time 10:** *"No More Alarms in Group(1): Display Thread <ThreadID> exiting at <TimeFromEpoch>."*

Actual Output:

```
Alarm> Start_Alarm(3): Group(1) 10 Testing

Alarm(3) Inserted by Main Thread 139884250270592 Into Alarm List at 1701818754: Group(1) 10
Testing

Main Thread Created New Display Alarm Thread 139884241872448 For Alarm(3) at 1701818754:
Group(1) 10 Testing

Alarm (3) Printed by Alarm Display Thread 139884241872448 at 1701818759: Group(1) 10 Testing

Alarm> Change_Alarm(3): Group(1) 10 new Alarm
Alarm Monitor Thread 139884250265152 Has Removed Alarm(3) at 1701818764: Group(1) 10
Testing

Change Alarm Request(3) Inserted by Main Thread 139884250270592 into Alarm List at 1701818764:
Group(1) 10 new Alarm
```

```
Invalid Change Alarm Request(3) at 1701818764: Group(1) 10 new Alarm

Display Thread 139884241872448 Has Stopped Printing Message of Alarm(3) at 1701818764:
Group(1) Testing

No More Alarms in Group(1): Display Thread 139884241872448 exiting at 1701818764.
```

Scenario 12: Testing Change Group ID.

Input:

- **Time 0:** *"Start_Alarm(3): Group(1) 10 Message"*
- **Time 10:** *"Change_Alarm(3): Group(2) 10 Message"*

Expected Output:

- **Time 0:** "Alarm(3) Inserted by Main Thread <ThreadID> Into Alarm List at <TimeFromEpoch>: Group(1) 10 Message"
- **Time 0:** "Main Thread Created New Display Alarm Thread <ThreadID> For Alarm(3) at <TimeFromEpoch>: Group(1) 10 Message"
- **Time 3:** "Change Alarm Request(3) Inserted by Main Thread <ThreadID> into Alarm List at <TimeFromEpoch>: Group(2) 10 Message"
- **Time 3:** "Alarm Monitor Thread <ThreadID> Has Changed Alarm(3) at <TimeFromEpoch>: Group(2) 10 Message"
- **Time 3:** "Main Thread Created New Display Alarm Thread <ThreadID> For Alarm(3) at <TimeFromEpoch>: Group(2) 10 Message"
- **Time 5:** "Display Thread <ThreadID> Has Stopped Printing Message of Alarm(3) at <TimeFromEpoch>: Changed Group(1) Message"
- **Time 5:** "No More Alarms in Group(1): Display Thread <ThreadID> exiting at <TimeFromEpoch>."
- **Time 8:** "Display Thread <ThreadID> Has Taken Over Printing Message of Alarm(3) at <TimeFromEpoch>: Changed Group(2) 10 Message"
- **Time 13:** "Alarm Monitor Thread <ThreadID> Has Removed Alarm(3) at <TimeFromEpoch>: Group(2) 10 Message"
- **Time 13:** "Display Thread <ThreadID> Has Stopped Printing Message of Alarm(3) at <TimeFromEpoch>: Group(2) Message"
- **Time 13:** "No More Alarms in Group(2): Display Thread <ThreadID> exiting at <TimeFromEpoch>."

Actual Output:

```
Alarm> Start_Alarm(3): Group(1) 10 Message

Alarm(3) Inserted by Main Thread 139735838620544 Into Alarm List at 1701820004: Group(1) 10
Message

Main Thread Created New Display Alarm Thread 139735830222400 For Alarm(3) at 1701820004:
Group(1) 10 Message
```

```
Alarm>Change_Alarm(3): Group(2) 10 Message

Change Alarm Request(3) Inserted by Main Thread 139735838620544 into Alarm List at 1701820007:
Group(2) 10 Message

Alarm>
Alarm Monitor Thread 139735838615104 Has Changed Alarm(3) at 1701820007: Group(2) 10
Message

Main Thread Created New Display Alarm Thread 139735754339904 For Alarm(3) at 1701820007:
Group(2) 10 Message

Display Thread 139735830222400 Has Stopped Printing Message of Alarm(3) at 1701820009:
Changed Group(1) Message

No More Alarms in Group(1): Display Thread 139735830222400 exiting at 1701820009.

Display Thread 139735754339904 Has Taken Over Printing Message of Alarm(3) at 1701820012:
Changed Group(2) 10 Message

Alarm Monitor Thread 139735838615104 Has Removed Alarm(3) at 1701820017: Group(2) 10
Message
Display Thread 139735754339904 Has Stopped Printing Message of Alarm(3) at 1701820017:
Group(2) Message

No More Alarms in Group(2): Display Thread 139735754339904 exiting at 1701820017.
```

6. Conclusion

In conclusion, the modifications to the "alarm_cond.c" program, resulting in "New_Alarm_Cond.c," introduce two types of alarm requests, "Start_Alarm" and "Change_Alarm," with specific formats. The main thread efficiently handles valid requests, inserting alarms into corresponding lists and dynamically creating display threads for distinct Group_IDs. The enhancements ensure a well-organized and responsive alarm management system with improved error handling and clear user communication. Overall, our system's synchronization relies on semaphores for efficient resource management and thread communication. Thread roles are well-defined: the main thread handles alarms and display assignments, the monitor manages changes and expiring alarms, while display threads print messages and monitor alarms. The monitor thread's efficiency lies in its dormant state, activating only when needed through timed waits on the monitor semaphore. Display threads update data and print messages, utilizing sleep intervals to minimize contention for shared resources. Reader and writer semaphores maintain consistency across threads. Functions like start_reading() and stop_reading() manage reader counts and semaphore access, ensuring a synchronized approach to resource handling. In essence, our synchronization strategy maximizes semaphore utilization, optimizes thread operations while preserving data integrity.

References

- Alarm_cond.c program sourced from "Programming with POSIX Threads" by David R. Butenhof