Programming Assignment 3 Report

Hüseyin Alper Karadeniz

Operating Systems (CS 307) / Fall 2022-2023

1 Program Flow and Pseudo Code

I used C programming language for this homework's implementation. The program flow and design choices of my program are explained in detail in this report. First of all, the program checks the validity of the given two inputs from the console, these are the number of fans from the Club A, and from the Club B. Total number of fans from both clubs must be divisible by 4, because we want to create bands of 4 people for each car, which is only possible when there is exactly fitting number of people. Also, both the number of fans of the Club A, and the number of fans of the Club B must be divisible by 2, since we don't want any bands which has 1 person from one club, but 3 from other club, there is only 4-0, 0-4, or 2-2 options for the distribution of the number of fans in each car.

After validations, two semaphores that had been created for the fans of the Club A, and for the Club B are initialized. Also, the barrier is initialized in order to maintain the condition of the bands of 4 people. Additionally, memory allocation operation is done in the total size of the threads, which will be created for maintaining the system. In my implementation, for each fan, a thread is being created with the function for her/his club. After the creation of all threads, they are joined with each other. Lastly, after joining operation, the program frees the dynamically allocated space for the threads and terminates with a termination message. Besides, if there is an error with the validation section in the beginning, the program again terminates with a termination message.

In my program implementation, there is a main function and also there are two functions for the fan threads. Fan thread functions are identical in operations, but use different variables since they are for the fans of two different clubs.

1.1 Main Function Flow & Pseudo Code:

- Number of fans from each club is taken from the user as a console input.
- Input validations are done according to the number of fans.

- Semaphores (one for the Club A + one for the Club B fans) are initialized.
- Thread barrier is initialized for the synchronization of threads.
- Threads (a unique thread for each fan) are executed.
    - A space on memory is dynamically allocated for all the threads.
    - Threads are created for the fans of Club A, with its specific function.
    - Threads are created for the fans of Club B, with its specific function.
    - All threads are joined with each other.
    - Dynamically allocated space for threads are freed up.
- The main terminates when the program ends, or if the inputs are invalid.

1.2 Thread Function for Club A Flow & Pseudo Code:

- The shared mutex is locked before the operations of the thread function.
- Initial print function, stating the fan is looking for a car, is executed.
- Number of fans waiting in the queue from Club A is incremented by one.
- Semaphores are adjusted according to the fans waiting in the queue.
    - If there are 4 A fans (the fans from Club A) now, this fan is captain.
        - Number of waiting fans from Club A is decremented by 4.
        - Semaphore post function for Club A is executed for four times.
    - If there are 2-3 A fans + more than 2 B fans now, this fan is captain.
        - Number of waiting fans from Club A is decremented by 2.
        - Number of waiting fans from Club B is decremented by 2.
        - Semaphore post function for Club A is executed for two times.
        - Semaphore post function for Club B is executed for two times.

- Else, this fan is not captain, so the shared mutex is unlocked.

- Semaphore of A waits for a signal for the formation of a car band.
- Second print function, stating the fan found a spot in a car, is executed.
- Barrier waits for the related fan threads to print their second messages.
- After barrier, it is checked if the current fan is the captain. If captain:
  - Third print function, stating the fan is the captain of car, is executed.
  - The shared mutex is unlocked now.

1.3 Thread Function for Club B Flow & Pseudo Code:

- The shared mutex is locked before the operations of the thread function.
- Initial print function, stating the fan is looking for a car, is executed.
- Number of fans waiting in the queue from Club B is incremented by one.
- Semaphores are adjusted according to the fans waiting in the queue.
  - If there are 4 B fans (the fans from Club B) now, this fan is captain.
    - Number of waiting fans from Club B is decremented by 4.
    - Semaphore post function for Club B is executed for four times.
  - If there are 2-3 B fans + more than 2 A fans now, this fan is captain.
    - Number of waiting fans from Club B is decremented by 2.
    - Number of waiting fans from Club A is decremented by 2.
    - Semaphore post function for Club B is executed for two times.
    - Semaphore post function for Club A is executed for two times.
  - Else, this fan is not captain, so the shared mutex is unlocked.
- Semaphore of B waits for a signal for the formation of a car band.

- Second print function, stating the fan found a spot in a car, is executed.
- Barrier waits for the related fan threads to print their second messages.
- After barrier, it is checked if the current fan is the captain. If captain:
  - Third print function, stating the fan is the captain of car, is executed.
  - The shared mutex is unlocked now.

2 Summary

My program implementation is correct, because there is no issues on synchronization or thread implementation, and it has no deadlock problems. There are some variables that are globally stored so that shared among the threads. These are the number of fans waiting in the queue from Club A and B, semaphores for the fans of Club A and B ensuring the correct formation of car bands, a mutex and a barrier preventing any other synchronization issues.

All of the fan threads and the main thread completes their executions by printing their dedicated lines to the console, they are joined to each other, so that the main thread waits for the fan threads to complete their executions, there is no deadlocks. Also, there is a correct ordering among the executions and outputs of the fan threads, which is ensured by the implementation of the shared mutex and the shared barrier.

In my program, there is no problem with one band, and no problem with multiple bands as well, because in the implementation, there is a loop for the creation of threads in the requested size, and after the validation of the inputs, main thread can create many fan threads with their dedicated functions without any problems. In addition to shared mutex and barrier, semaphores will ensure that the all threads are running concurrently without any synchronization issues.