Ege Aktemur 28080

19/12/2022

Programming Assignment 3 Report

In this report I will firstly explain my program flow. Then I will go over Synchronization Primitives I implemented and solutions that I used.

Program Flow

First get the given count of A team fans amount and B team fans amount. If the given amounts are not valid terminate the program. Else create a random vector with given A and B fans in it. Then initialize the semaphores, barrier and global variables and create threads for each fan.

Secondly in the function get the team as a char (A or B) and initialize the variables to read the global variables. If the function already has enough fans (checked using the full semaphore), wait until the function needs another fan, else print the thread id and team info.

After the thread passes the full semaphore and prints it will get the semaphore and count information from the global variables to its variables. After that I lock until it processes the data and it does the necessary checks and operations.

These checks and operations are:

1. Increase the current team's count by 1.
2. Check if this fan is waiting with 4 other fans. In this case it should not get new fans when exiting.

3. A fan should check these 4 conditions:
    3.1. If it is a 4-0 or 4-1 situation:
        3.1.1. Set this fan as the captain.
        3.1.2. Decrease the current team amount by 4.
        3.1.3. Awake 3 current team members.
    3.2. If it is 2-2 or 2-3 situation:
        3.2.1. Set this fan as the captain.
        3.2.2. Decrease the current team amount and other team amount by 2.
        3.2.3. Awake 1 current team and 2 other team members.
    3.3. If we could not form a group but we have already 4 fans (3-1 or 1-3 situation):
        3.3.1. Get one more fan.
        3.3.2. Sleep until a group forms.
    3.4. If we could not form a group and we do not have already 4 fans (Rest of the situations):
        3.4.1. Sleep until a group forms.

After all threads of the care are awake they should announce that they found a car with their thread ids and team info.

They must wait for other threads to print for the captain to print he/she is the captain because we do not want the captain to print before there are fans that have not printed yet..

Then again they should wait for the captain to get new fans because we do not want the new fan to print between finding a car and the captain message.

After all the printing is complete all fans that should post (except the threads that joined as the 5th fan), post and join to main.

Lastly in the main wait the threads that have not joined yet to join.

Pseudocodes

Algorithm 1 Function For threads

Input team character, team A and B waiting counts, A , B, full semaphores, print and exec mutexes, wait barrier
Output None

```
1: procedure FUNC(team)
2:     Initialize variables
3:     While print is locked print I am looking for a car info
4:     if team == 'A' then
5:         Get and set global semaphores and team amounts
6:     else
7:         Get and set global semaphores and team amounts
8:     end if
9:     Lock execute mutex
10:    increment currentTeamAmount by 1
11:    If Total Amount is 5 set should_post to False
12:    if currentTeamAmount == 4 then
13:        Set captain to True
14:        Decrement Current Team Amount by 4
15:        Awake 3 Current Team Members
16:    else if currentTeamAmount == 2 and otherTeamAmount ≥ 2 then
17:        set captain to True
18:        Decrement Current Team Amount by 1 and Other Team Amount by 2
19:        Awake 1 Current Team Member and 2 Other Team Members
20:        Unlock execute mutex
21:    else if currentTeamAmount + otherTeamAmount == 4 then
22:        Get one more fan
23:        Unlock execute mutex
24:        Sleep until awaken
25:    else
26:        Unlock execute mutex
27:        Sleep until awaken
28:    end if
29:    While print is locked print I have found a spot in a car info
30:    Wait until all 4 fan threads arrive
31:    If fan is the captain while print is locked print I am the captain info
32:    Wait until all 4 fan threads arrive
33:    fan should post get one more fan
34: end procedure
```

Algorithm 2 Main

Input command line arguments for counts of A and B fans
Output None

```
1: procedure MAIN(argc, argv)
2:     Get A and B count and set CountTotal
3:     If given counts are not correct exit the program
4:     Create Random Vector with A and B's according to counts
5:     Initialize semaphores, barriers
6:     for i in range(0, countTotal) do
7:         create a new thread with function func and argument RandTotalFans[i]
8:     end for
9:     join all threads
10: end procedure
```

Synchronization Primitives

- Semaphore A / Semaphore B
    - Semaphores used to wake(when a car can not be formed) and awake (when a car can be formed) A/B team fan threads.
- Semaphore Full
    - Semaphore used to get a new fan to the function if it is needed.
    - This semaphore guarantees that there will always be 4 fans in the car.
    - This Semaphore is initialized with value 4 and every time a fan gets in the function it will wait if value is 0 (4 fans in the function). Else it will decrease the value by 1 and do the operations.
    - Later when exiting the function it will post by 1 to get a fan instead of itself.
    - However, as I explained, if there are already 4 fans and this fan is the 5th (3-1 or 1-3 case) this fan should not post for a new fan (value is 3 at the end) because there is already a fan waiting.
- Mutex Print
    - Used for synchronization of print statements. (To block print statements to interrupt other print statements and guarantee print concurrency)
- Mutex Execution
    - Used for variable concurrency. For example if a fan had 3 same team fans waiting and incremented it's team amount and before checking the current team amount is 4 another fan may come and increase team amount and we would not be able to form a group of 4.
- Barrier Wait
    - As I explained this barrier makes sure for the captain to start and finish at the correct time.