CS307 - FALL 23-24 - PROGRAMMING ASSIGNMENT 3

Riding to a Soccer Game

AHMET ALPEREN GÜNGÖR

28847

- In programming assignment 3, the main goal is to design an algorithm for Unix command line based rideshare application where fans of 2 opposing soccer teams can share a care to the stadium - according to some predetermined rules. In order to implement it properly, multiple synchronization mechanisms are used:

    Mutex: A pthread mutex called global lock is used to ensure critical sections of thread function (create_rider) are not disturbed.

    Semaphores: 2 semaphores called sem_A for team A's fans and sem_B for team B's fans are used in order to implement predetermined rules for rideshare – negative values represent the amount of fans of team A or B waiting.

    Barrier: A pthread barrier is used to ensure synchronization of threads for a formed car to print to the console properly.

  For the rules of rideshare, the following algorithm is implemented:

    1. if the current thread's team's number is 0, then wait

    2. if the current thread's team's number is 1

        a. if other team's number is less than 2 (it can be 0 or 1), then wait

        b. else (it can be 2 or 3 - it cannot be 4 or more since they would already form a car), then form a car and be the captain

    3. if the current thread's team's number is 2, then wait

    4. if the current thread's team's number is 3, then form a car and be the captain

  If the proper conditions to form a team are not satisfied, semaphores ensures those threads are sleeping – while providing information on how many of them are sleeping.

- Program Flow: First, the number of fans for each team is taken as input from the command line. If the number are not even for teams – or if the total number of funs are not multiple of 4, program terminates. After, synchronization variables are initialized, a thread list for threads are created. For each fan, a thread is created using for loops – which calls the create_rider function.

In this function, a Boolean value is stored - which is set as true when a car is formed. A global lock is used to ensure shared variables do not conflict with other threads. Then, the algorithm explained above is implemented. In this implementation, captain is chosen as the last thread that creates the car, therefore if a thread cannot form a team, it waits until someone else can. When a car is formed, proper amount of threads are waken up by sem_post, which are then proceed to wait in a barrier to ensure correct order of printing to the console. Since the car is formed, Boolean variables is set to true, id is increased by 1, barrier is destroyed and global lock is removed. All these synchronization variables ensure correct ordering of outputs and flow of the program.

- A pseudocode for a sample program flow can be as:

1. Input to the command line 2 6 – which means 2 fans of team A and 2 fans of team B.
2. Condition check passed
3. Synch variables are initialized (sem initialized with val 0 since neg num = fans waiting)
4. Correct amount of threads are created for each team's fans via 2 for loops – which call create_rider function:
    a. Thread of team a enters function, prints looking for team, since val for sem_A is currently 0, waits.
    b. Thread of team b enters function, prints looking for team, since val for sem_B is currently 0, waits.
    c. Thread of team a enters function, prints looking for team, val for sem_A is -1 but for sem_B it is not -2 therefore waits.
    d. Thread of team b enters function, prints looking for team, val for sem_B is -1 and sem_A is -2, therefore a team can be formed. Makes the bool variable true, wakes 1 team B thread, wakes 2 team A threads.
    e. All car member threads print car spot found and waits at barrier. Since bool is check as true (for the captain), if condition entered, captain prints his and car's id, increases the car id by 1, destroys barrier (and initializes again), unlocks the global lock.
    f. For the rest of the 4 threads of B, first 3 wait (sem_wait) until 4^{th} thread runs, 4^{th} one enters the if condition since sem_B.val is -3, wakes other 3 members of car and checks bool as true, all of them print found a car, captain enters if condition, prints his and teams id, increases car id by 1, destroys barrier (and initializes again), unlocks the global lock.
5. All threads are finished, their join is waited via a for loop in main, then main terminates.