Yiğit Karamanlı

28105

yigitkaramanli@sabanciuniv.edu

PSEUDO-CODE OF THE THREADS

Get the team info from the arguments and print “looking for a car”.

Acquire the global lock

Check if any of the two conditions are satisfied.

    If not, release the global lock and start sleeping,

    If satisfied, wake necessary combination of threads and declare itself as the captain, not output it yet.

Print “found a spot”

Wait at the barrier

If it is the captain

    Print captain message

    Destroy and reinitialize the barrier

Terminate/return

SYNCHRONIZATION MECHANISMS USED

I have implemented my own semaphores using pthread mutexes and condition variables with the help of the implementation in the lecture slides of the synchronization chapter and used one semaphore for each Team, semA and semB respectively. I have changed the sem_wait() function and added a mutex_ unlock statement to unlock the global mutex after decrementing the value. Since each thread checks the value of the semaphore in order to control if the conditions are satisfied or not, this makes it impossible for threads to decrement the value nearly at the same time due to context switches and prevents potential deadlocks that may be caused due to all threads sleeping with no thread to wake them up.

I have used a barrier to make it sure that each thread woken up by the thread prints that it has found a spot before the captain reveals itself by printing to the console “I am the captain”., thus satisfying one of the correctness criteria described in the assignment document. The other correctness criteria are satisfied by my usage of the pthread mutexes and semaphores.

IMPLEMENTATION DETAILS

Semaphore implementation

```
typedef struct semaphore   // Implemented
{
    int val;   // Value of the semaphore
    pthread_mutex_t lock; // lock
    pthread_cond_t c; // cond. variable
} sem_t;
```

sem_ wait() and sem_post()

sem_lockandwait() function has pthread_mutex_unlock() to unlock global mutex after the decrement operation in order to prevent a deadlock that may be caused from threads sleeping again right after they wake up.

```
void sem_lockandwait (sem_t *s, pthread_mutex_t * lock) { // se
// decrement the value of s by one
// wait if the value of s is negative
pthread_mutex_lock(&s->lock);
s->val --;                      //decrement the value of the se
pthread_mutex_unlock(lock);     //unlock the global lock here,
if (s->val < 0)
pthread_cond_wait(&s->c, &s->lock);
pthread_mutex_unlock(&s->lock);
}

void sem_post (sem_t *s) {  // sem_post() from lecture slides
// increment the value of s by one
// wake a sleeping thread if exists
pthread_mutex_lock(&s->lock);
s->val ++;
pthread_cond_signal(&s->c);
pthread_mutex_unlock(&s->lock);
}
```

Checking of conditions inside the thread function: this implementation makes it sure that the last thread satisfying the condition will wake the other threads.

```
pthread_mutex_lock(&globlock); // acquire the mutex
//printf("Thread ID: %lu controlling\n", pthread_self());  // used du
if(sem_val(fan->sem1) <= -1 && sem_val(fan->sem2) <= -2){   //check i
    Cap = true;  // set thread as the captain
    sem_post(fan->sem1);
    sem_post(fan->sem2);     //wake up 2 rival threads and 1 with the
    sem_post(fan->sem2);
}
else if (sem_val(fan->sem1)== -3) // if threre are 3 more threads wai
{
    Cap = true;                         // set thread as captain
    sem_post(fan->sem1);
    sem_post(fan->sem1);                //wake up the 3 threads
    sem_post(fan->sem1);
}
else{                  // if the conditions are not met.
    //printf("Thread ID: %lu unlocked\n", pthread_self());    // for
    //printf("Thread ID: %lu zzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzz
    sem_lockandwait(fan->sem1, &globlock);    //decrement the value c
}
```