CSE-344 SYSTEM PROGRRAMING

SPRING 2024

HOMEWORK 3

REPORT

ONUR ATASEVER

210104004087

Problem:

There is a parking area for vehicles to park. Vehicle owners park their vehicles in the temporary parking area (if not occupied) and valets park the vehicles in the main parking area.

In the system, there are 12 spaces in total in the temporary parking area, 4 for pickup vehicles and 8 for cars. Only one vehicle can enter the temporary parking area at a time. If the temporary parking area for the vehicle type is full, the owner leaves with his vehicle.

There are 2 valets in the system. Both valets can park pick-ups and cars in the main parking area, but only one type of vehicle can be parked at a time. For example, if the first valet picks up and parks the car, the second valet must park the car. Or vice versa. And there are infinite spaces in the main park area.

Problem Solving Approach:

The main purpose is to provide the necessary protection in critical areas and to ensure synchronization between threads.

I created 4 different threads as the main part of the problem. One thread was created for each vale and one thread was created for each vehicle-creating function.

```
// Create attendant threads
pthread_create(&auto_attendant_thread, NULL, carAttendant, NULL);
pthread_create(&pickup_attendant_thread, NULL, carAttendant, NULL);

// Create vehicle generator threads
pthread_create(&pickup_gen_thread, NULL, pickup_generator, NULL);
pthread_create(&automobile_gen_thread, NULL, automobile_generator, NULL);
```

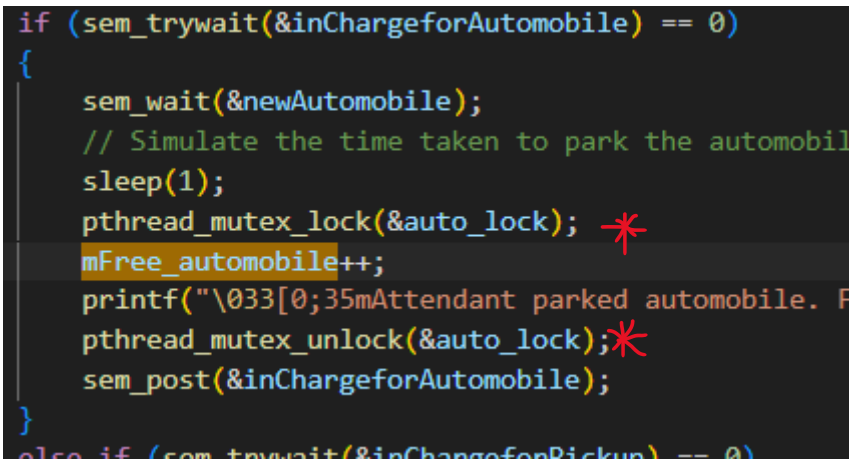
Critical areas: The parts where the counter values (mFree_automobile, mFree_pickup) for both vehicle types are changed, the part where vehicles enter in the carOwner function (one vehicle can enter at the same time).

Changing counter values:

If there is space in the temporary parking area, the vehicle settles in the temporary parking area and reduces the remaining space by one. At the same time, if the valet takes a vehicle from the temporary parking area and parks it in the main parking area, the number of remaining parking spaces increases by one.

I used mutex to ensure synchronization of events occurring in these two different threads and affecting the same variable and to prevent them from changing the variable at the same time.

```
if (vehicle_type == 'A')
{
    pthread_mutex_lock(&auto_lock);
    if (mFree_automobile > 0)
    {
        mFree_automobile--;
        printf("\033[0;36mAutomobile owner parked in temporary spot.
        sem_post(&newAutomobile);
    }
    else
    {
        printf("\033[0;31mAutomobile owner left, no free temporary s
        pthread_mutex_unlock(&auto_lock);
        pthread_mutex_unlock(&entry_lock);
        return NULL;
    }
    pthread_mutex_unlock(&auto_lock);
}
```



```
if (sem_trywait(&inChargeforAutomobile) == 0)
{
    sem_wait(&newAutomobile);
    // Simulate the time taken to park the automobil
    sleep(1);
    pthread_mutex_lock(&auto_lock);
    mFree_automobile++;
    printf("\033[0;35mAttendant parked automobile. F
    pthread_mutex_unlock(&auto_lock);
    sem_post(&inChargeforAutomobile);
}
```

As can be seen in the examples, the increment and decrement operations (critical regions) for the automobile free space counter are controlled by mutex and thus, only one operation is performed on that variable at a time. The same applies to pick-up style vehicles.

When the car generator and pick up generator creates a vehicle, it automatically calls the carOwner function. If these generators in two different threads create vehicles at the same time, the vehicles will want to enter the system at the same time. However, only one vehicle can enter the system at a time. A mutex was used to prevent the entry of a vehicle in the other thread and wait for a vehicle to enter.

```
void *carOwner(void *arg)
{
    char vehicle_type = *(char *)arg;

    pthread_mutex_lock(&entry_lock); // Ensure only one vehicle can enter at a time

    if (vehicle_type == 'A')
    {
```

```
    }

    pthread_mutex_unlock(&entry_lock); // Release the entry lock
    return NULL;
}
```

Mainly, 4 different semaphores were used for thread synchronization. newAutomobile makes sem_post when the vehicle enters the system (if there is space). On the other side, the valet responsible for parking the car is waiting for the car with sem_wait. If this value is greater than 0 (if sem_post has been made before), it means there is a car. Thus, the parking process continues. However, if the value in the sem_wait function is 0, it means there is no vehicle. The valet waits until a vehicle arrives and increases its value with the sem_post function. The same goes for the newPickup semaphore. The valet waits with sem_wait to park. When the pick-up style vehicle arrives, the value is increased with Sem_post and the valet stops waiting with sem_wait and then parking is done. Thus, synchronization of the presence of vehicles and parking of valets is ensured.

```
if (mFree_automobile > 0)
{
    mFree_automobile--;
    printf("\033[0;36mAutomobile owner parked in temporary spot. Free automobile spots: %d\n", mFree_automobile);
    sem_post(&newAutomobile);
}
```

```
if (sem_trywait(&inChargeforAutomobile) == 0)
{
    sem_wait(&newAutomobile);
    // Simulate the time taken to park the automobile
    sleep(1);
    pthread_mutex_lock(&auto_lock);
    mFree_automobile++;
    printf("\033[0;35mAttendant parked automobile. Free automobile spots: %d\n", mFree_automobile);
    pthread_mutex_unlock(&auto_lock);
    sem_post(&inChargeforAutomobile);
}
```

Two other semaphores used for synchronization are inChargeforAutomobile and inChargeforPickup. These two semaphores are used to ensure synchronization when valets park their vehicles from the temporary parking area to the main parking area. Each valet can park both types of vehicles, but only one type of vehicle can be parked at a time. So, if the first valet picks up and parks the car, the second valet must park the car. Or vice versa.

```
if (sem_trywait(&inChargeforAutomobile) == 0)
{
    sem_wait(&newAutomobile);
    // Simulate the time taken to park the automobile
    sleep(1);
    pthread_mutex_lock(&auto_lock);
    mFree_automobile++;
    printf("\033[0;35mAttendant parked automobile. Free automobile spots: %d\n", mFree_automobile);
    pthread_mutex_unlock(&auto_lock);
    sem_post(&inChargeforAutomobile);
}
else if (sem_trywait(&inChargeforPickup) == 0)
{
    sem_wait(&newPickup);
    // Simulate the time taken to park the pickup
    sleep(1);
    pthread_mutex_lock(&pickup_lock);
    mFree_pickup++;
    printf("\033[0;35mAttendant parked pickup. Free pickup spots: %d\n", mFree_pickup);
    pthread_mutex_unlock(&pickup_lock);
    sem_post(&inChargeforPickup);
}
```

The sem_trywait(&inChargeforAutomobile) function attempts to lock the inChargeforAutomobile semaphore. If the semaphore is already locked, sem_trywait returns -1 and the code moves on to the next else if block (It means other vale is parking an automobile and currenct vale has to park a pick up). If the semaphore is not locked, sem_trywait locks it and returns 0 (It means other vale is not parking an automobile and current vale can park pick up.), and the code inside the if block is executed.

The sem_post(&inChargeforAutomobile) function signals the inChargeforAutomobile semaphore to indicate that the attendant is done parking the automobile.

```c
void *pickup_generator(void *arg)
{
    char vehicle_type = 'P';
    while (running)
    {
        carOwner(&vehicle_type);
        usleep((rand() % 1000 + 100) * 1000); // Sleep for 100ms to 1s
    }
    return NULL;
}

// This function simulates a vehicle generator that generates vehicles at random intervals.
void *automobile_generator(void *arg)
{
    char vehicle_type = 'A';
    while (running)
    {
        carOwner(&vehicle_type);
        usleep((rand() % 1000 + 100) * 1000); // Sleep for 100ms to 1s
    }
    return NULL;
}
```

These functions are seperated threads. They generates new vehicles and they calls carOwner function and in the carOwner function necessary ciritical section protections and semaphores are done.

To handle CTRL + C and exit program

```c
struct sigaction sa;
sa.sa_handler = intHandler;
sigemptyset(&sa.sa_mask);

sa.sa_flags = SA_RESTART;

if (sigaction(SIGINT, &sa, NULL) == -1)
{
    perror("sigaction");
    exit(EXIT_FAILURE);
}
```

To finish program you shoduld interrup the program with CTRL+C. Signal handler which is inthandler, destroy all the semaphores and joins threads. And then it exits succesfully.

Output example:

```
Pickup owner parked in temporary spot. Free pickup spots: 3
Automobile owner parked in temporary spot. Free automobile spots: 7
Automobile owner parked in temporary spot. Free automobile spots: 6
Automobile owner parked in temporary spot. Free automobile spots: 5
Attendant parked automobile. Free automobile spots: 6
Attendant parked pickup. Free pickup spots: 4
Pickup owner parked in temporary spot. Free pickup spots: 3
Automobile owner parked in temporary spot. Free automobile spots: 5
Pickup owner parked in temporary spot. Free pickup spots: 2
Pickup owner parked in temporary spot. Free pickup spots: 1
Attendant parked automobile. Free automobile spots: 6
Automobile owner parked in temporary spot. Free automobile spots: 5
Attendant parked pickup. Free pickup spots: 2
Pickup owner parked in temporary spot. Free pickup spots: 1
Automobile owner parked in temporary spot. Free automobile spots: 4
Automobile owner parked in temporary spot. Free automobile spots: 3
Attendant parked automobile. Free automobile spots: 4
Attendant parked pickup. Free pickup spots: 2
Pickup owner parked in temporary spot. Free pickup spots: 1
Automobile owner parked in temporary spot. Free automobile spots: 3
Pickup owner parked in temporary spot. Free pickup spots: 0
Attendant parked automobile. Free automobile spots: 4
Attendant parked pickup. Free pickup spots: 1
Pickup owner parked in temporary spot. Free pickup spots: 0
Automobile owner parked in temporary spot. Free automobile spots: 3
Automobile owner parked in temporary spot. Free automobile spots: 2
Pickup owner left, no free temporary spot.
Automobile owner parked in temporary spot. Free automobile spots: 1
Pickup owner left, no free temporary spot.
Attendant parked automobile. Free automobile spots: 2
Attendant parked pickup. Free pickup spots: 1
Automobile owner parked in temporary spot. Free automobile spots: 1
Pickup owner parked in temporary spot. Free pickup spots: 0
Attendant parked automobile. Free automobile spots: 2
Attendant parked pickup. Free pickup spots: 1
Automobile owner parked in temporary spot. Free automobile spots: 1
Automobile owner parked in temporary spot. Free automobile spots: 0
Pickup owner parked in temporary spot. Free pickup spots: 0
Automobile owner left, no free temporary spot.
Attendant parked automobile. Free automobile spots: 1
Attendant parked pickup. Free pickup spots: 1
Pickup owner parked in temporary spot. Free pickup spots: 0
Automobile owner parked in temporary spot. Free automobile spots: 0
Attendant parked automobile. Free automobile spots: 1
Attendant parked pickup. Free pickup spots: 1
Pickup owner parked in temporary spot. Free pickup spots: 0
^CAttendant parked automobile. Free automobile spots: 2
Attendant parked pickup. Free pickup spots: 1
Simulation ended by user.
```

Output Example 2:

```
Automobile owner parked in temporary spot. Free automobile spots: 7
Pickup owner parked in temporary spot. Free pickup spots: 3
Automobile owner parked in temporary spot. Free automobile spots: 6
Pickup owner parked in temporary spot. Free pickup spots: 2
Attendant parked automobile. Free automobile spots: 7
Attendant parked pickup. Free pickup spots: 3
Automobile owner parked in temporary spot. Free automobile spots: 6
Pickup owner parked in temporary spot. Free pickup spots: 2
Automobile owner parked in temporary spot. Free automobile spots: 5
Attendant parked automobile. Free automobile spots: 6
Attendant parked pickup. Free pickup spots: 3
Pickup owner parked in temporary spot. Free pickup spots: 2
Automobile owner parked in temporary spot. Free automobile spots: 5
Attendant parked automobile. Free automobile spots: 6
Attendant parked pickup. Free pickup spots: 3
Pickup owner parked in temporary spot. Free pickup spots: 2
Pickup owner parked in temporary spot. Free pickup spots: 1
Automobile owner parked in temporary spot. Free automobile spots: 5
Pickup owner parked in temporary spot. Free pickup spots: 0
Attendant parked automobile. Free automobile spots: 6
Attendant parked pickup. Free pickup spots: 1
Pickup owner parked in temporary spot. Free pickup spots: 0
Automobile owner parked in temporary spot. Free automobile spots: 5
Pickup owner left, no free temporary spot.
^CAttendant parked automobile. Free automobile spots: 6
Attendant parked pickup. Free pickup spots: 1
Simulation ended by user.
```

Memory leak checks:

```
==25686==
==25686== HEAP SUMMARY:
==25686==     in use at exit: 0 bytes in 0 blocks
==25686==   total heap usage: 5 allocs, 5 frees, 2,112 bytes allocated
==25686==
==25686== All heap blocks were freed -- no leaks are possible
==25686==
==25686== For lists of detected and suppressed errors, rerun with: -s
==25686== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
root@DESKTOP-PLJPMRB:/home/344-hw3#
```