GEBZE TECHNICAL UNIVERSITY

CSE 344
SYSTEM PROGRAMMING
Homework #3 Report

OGUZ MUTLU
1801042624

1.Problem Definition

Two parking attendants supervise a parking lot with a capacity of four places for pickups and eight places for automobiles. Various vehicles, such as automobiles and pickups, arrive at the parking lot entrance.

Requirements:

- The arrival of vehicles to the parking lot should be generated by a random number, odd:car, even:pickup.

-Each parking assistant must be assigned to one vehicle, one for the pickup and one for the car (2 threads for just parking asistant **carAttendant_auto**, **carAttendant_pickup**).

-Vehicle owners will come to the parking lot one by one.(Car owner thread)

-These three threads are synchronized with 4 semaphores.(**newPickup**, **inChargeforPickup**, **newAutomobile**, **inChargeforAutomobile**)

-There is a maximum of 8 cars and 4 pickups, parking space.(**mFree_automobile**, **mFree_Pickup**)

-If the parking areas are full, an evacuation mechanism must be provided.

-Owners must confirm their vehicle's parking availability.

-The occupancy of the temporary parking area and vehicle types are handled correctly.

-Only one vehicle can enter the system at a time.

2.Implementation Details

First of all, I followed the following approach to solve this problem; I started by creating 3 threads for owner, car parking assistant and pickup assistant.

After creating 3 threads with pthread_create, when these threads enter, the while loop starts to run. This while loop is controlled with the status value. The while loop ends only when the SIGINT signal arrives (while(status)). Synchronization must be ensured when the while loop starts in each thread.

SIGINT treats **runningStatus** variable as 0 and terminate threads.

All semaphores are initialized with 0.The program flow is provided as follows;

Owner Thread

-When the **CarOwner** thread starts executing, it generates a random number

-if this number is even

*First of all, the owner checks whether there are enough suitable spaces in the parking lot by looking at the mFree variables.

* The newAutomobile semaphore is posted and the log record that the new car has entered the system is printed on the screen.

*inChargeForAutomobile semaphore goes into wait state (the thread is blocked because it starts with 0)

*By posting the inChargeAutomobile semaphore in the carAttendant_automobile thread, the execution is restarted for random number generation.

-if this number is odd

*First of all, the owner checks whether there are enough suitable spaces in the parking lot by looking at the mFree variables.

* The newPickup semaphore is posted and the log record that the new car has entered the system is printed on the screen.

*inChargeForPickup semaphore goes into wait state (the thread is blocked because it starts with 0)

*By posting the inChargePickup semaphore in the carAttendant_pickup thread, the execution is restarted for random number generation.

After executions are done, I generate new random number for leaving system my leaving system is like rand() mod 11 = 0 then one car have to leave if odd pickup leaves if even automobile leaves, I choose 11 because mod is barely happen. After that system sleep(5) second for user friendly execution

CarAttendantAutomobile Thread

```
while(runStatus)
{
    sem_wait(newAutomobile);
        if(mFree_automobile > 0)
            mFree_automobile--;
        numBytesWrittenCarOwner = snprintf(log, BUFF_SIZE, "Availab
        write(STDOUT_FILENO, log, numBytesWrittenCarOwner);
        memset(log, 0, numBytesWrittenCarOwner);
    sem_post(inChargeforAutomobile);
}
```

-When the **carAttendantAutomobile** thread starts execution, the **newAutomobile** semaphore starts with a wait, and since it is initialized with 0, the semaphore thread is blocked during this wait.

-When **newAutomobile** is posted by the owner, this thread blocking condition is removed and the execution continues, reducing the number of parking spaces by one. and posts the **inChargeForAutomobil** semaphore, allowing the owner thread to continue execution.

carAttendantPickup Thread

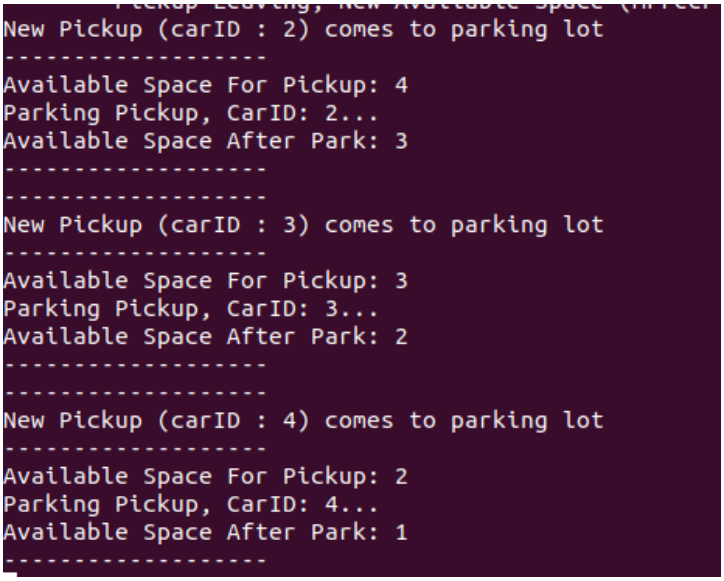
```
while(runStatus)
{
    sem_wait(newPickup);
        if(mFree_pickup > 0)
            mFree_pickup--;
        numBytesWrittenCarOwner = snprintf(log, BUFF_SIZE, "------
        write(STDOUT_FILENO, log, numBytesWrittenCarOwner);
        memset(log, 0, numBytesWrittenCarOwner);
    sem_post(inChargeforPickup);
}
```

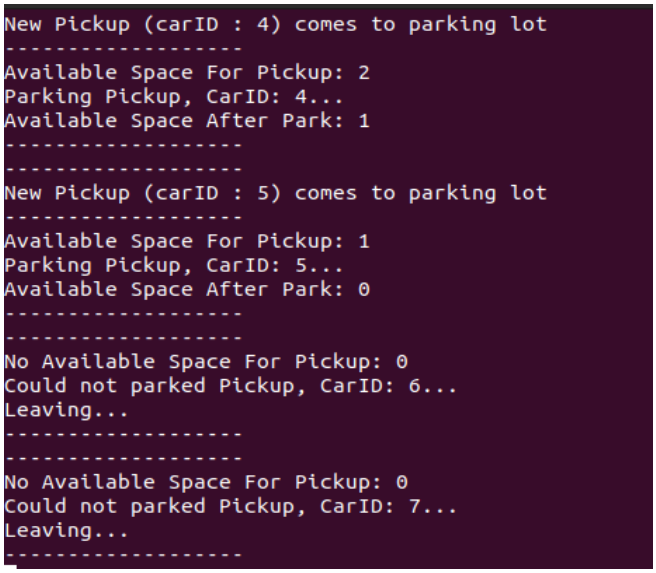
-When the **carAttendantPickup** thread starts execution, the **newPickup** semaphore starts with a wait, and since it is initialized with 0, the semaphore thread is blocked during this wait.

-When **newPickup** is posted by the owner, this thread blocking condition is removed and the execution continues, reducing the number of parking spaces by one. and posts the **inChargeForPickup** semaphore, allowing the owner thread to continue execution.

3.Test Cases

3.1. Scenario where only pickup arrives



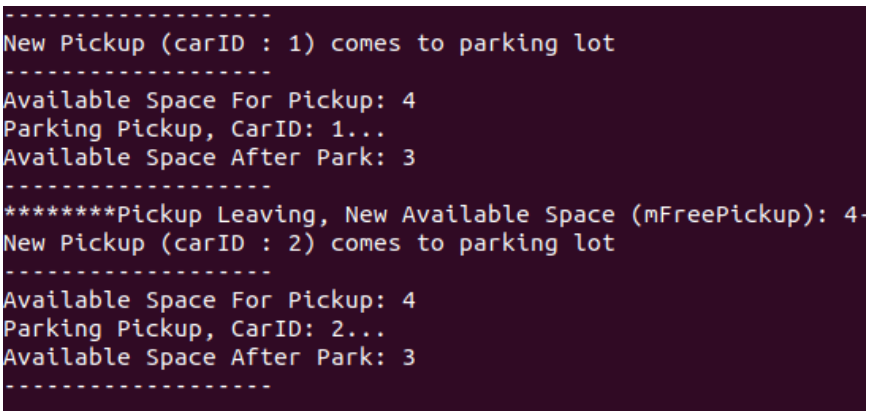


As can be seen in above, the number of free parking spaces defined as available space decreases only when the pickup arrives.And Leaving status can be shown

-pickup arriving ***(Successfull)***

-No available Space for Pickup ***(Successfull)***

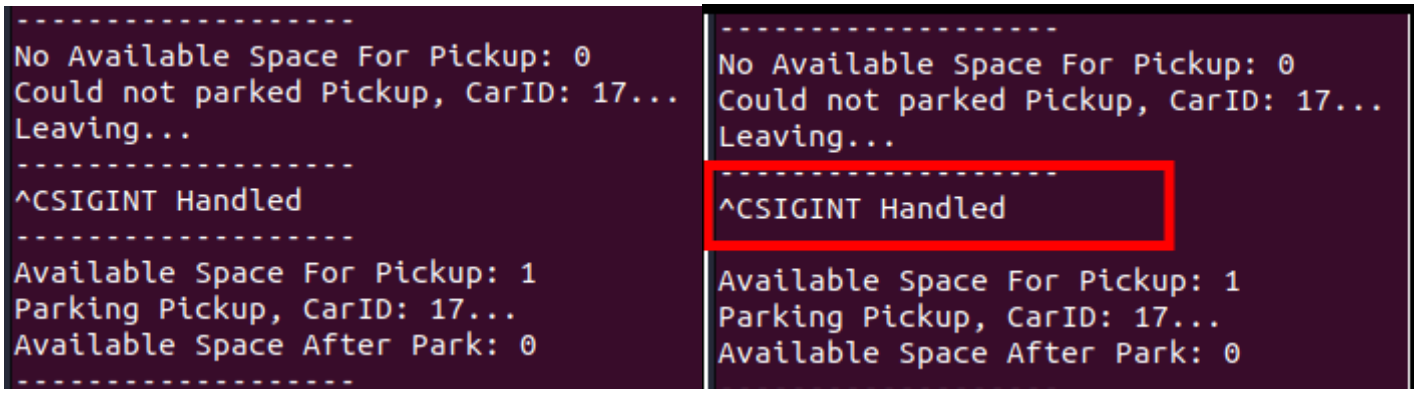
3.2 Leaving Mechanism For pickup



As you can see in the test experiment above, after the pick up park decreases to 3(2nd figure 1st box), a pickup leaves(2nd figure 3rd box) and the free number increases again by one(2nd figure 2nd box).

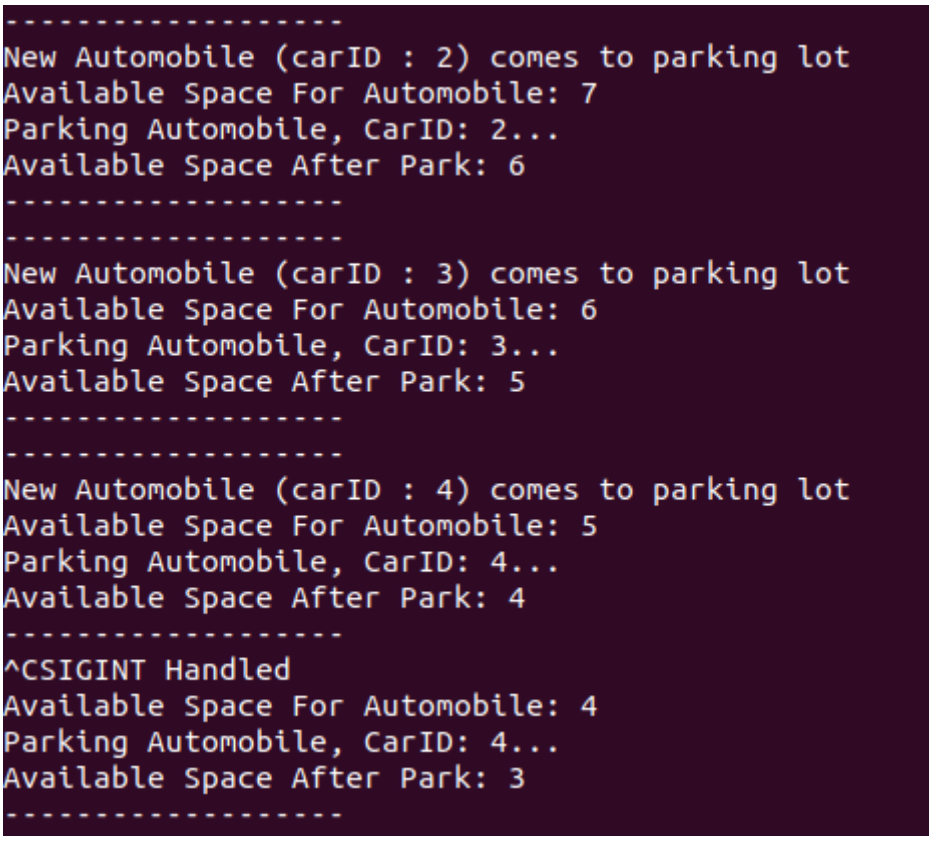
Leaving for Pickup-decrease and increase ***(Successfull)***

3.3 Program Termination



Program termination with sigint **(Successfull)**

3.4 Scenario where only Automobile arrives

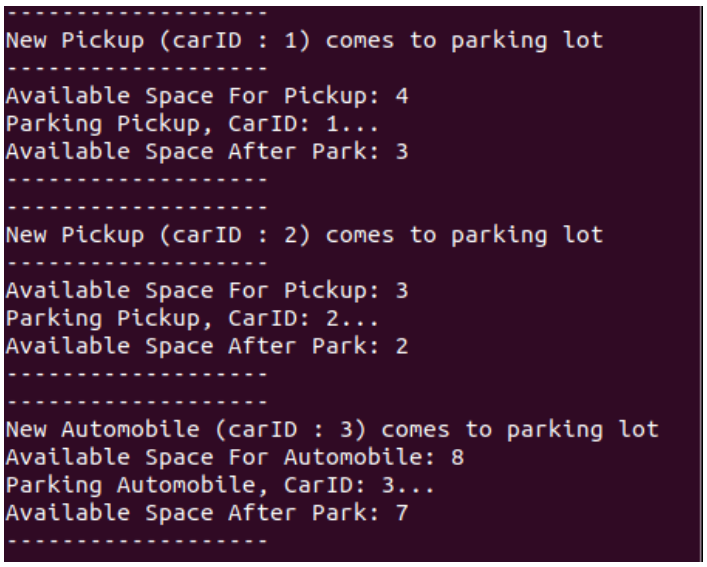


As can be seen in above, the number of free parking spaces defined as available space decreases only when the automobile arrives.

-Automobile arriving ***(Successfull)***

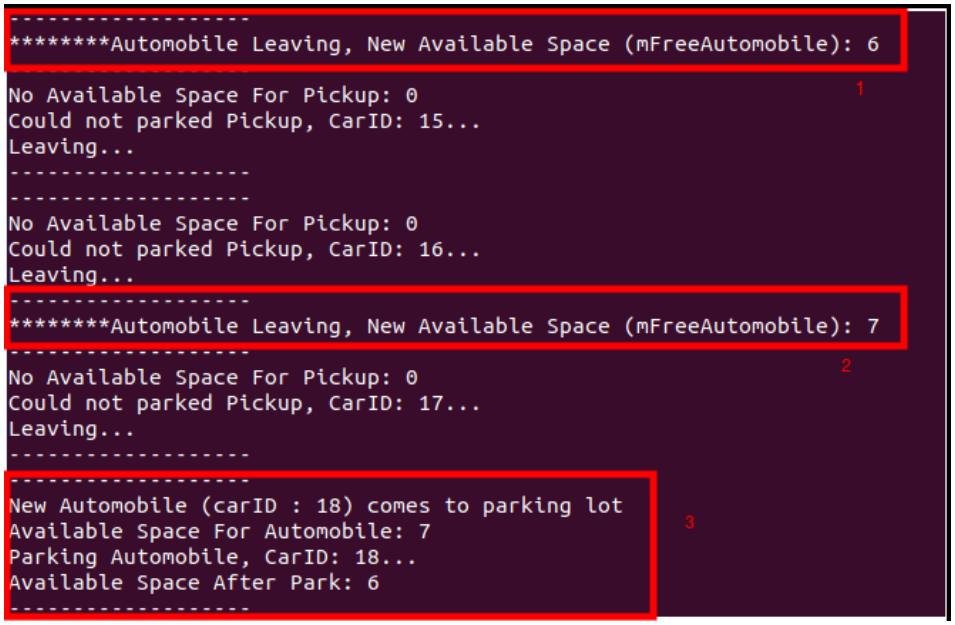
- *No available space for automobile* ***(Successfull)*** **shown in below*

3.5 Full System Test



As can be seen in the figure above, the generation of cars and pickups at certain intervals and their placement in the parking system are seen at certain intervals in order to be read user friendly.

In the continuation of the program, as seen in the test case above, it seems that the pickup parking area is finished, there is still space for cars until the 22nd vehicle, and it also appears that one pickup will be reserved and its place will be filled immediately.



In the flow above, the scenario of a car leaving the park, another car leaving the car without being parked on top of it, and then a car arriving at the parking area is shown. The system has accomplished this successfully.

3.6 Memory Leak Check

```
==21895==
==21895== HEAP SUMMARY:
==21895==     in use at exit: 0 bytes in 0 blocks
==21895==   total heap usage: 11 allocs, 11 frees, 1,124 bytes allocated
==21895==
==21895== All heap blocks were freed -- no leaks are possible
==21895==
==21895== For lists of detected and suppressed errors, rerun with: -s
==21895== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

>> valgrind ./hw3

leak check – **Successfull**

NOTE TO TA: Compile warnings do not affect the execution of the program, they show it that way because the argument is given to the thread and I have to initialize as that way.

The warning about not reaching the end of the while loop thread is also solved with SIGINT signal, and that warning does not pose any problem.

Kind regards.