CS 120

Programming Assignment 3

Important Note!

Unlike previous assignments, you need to modify two files and your submission will contain TWO files: mycode3.c and pa3d.c

- your modifications to the operating system in mycode3.c
- your solution to Exercise D contained in pa3d.c

The command to make submission is still same. Only thing is, you have to make sure both mycode3.c and pa3d.c files in **pa3 folder** are proper before making submission.

Important Note!

Also this assignment will be harder to debug- bugs will be difficult to reproduce.

If your implementation have some bugs, even then sometimes your code will work. And it will be difficult to reproduce the same error.

Where to start the assignment from ?

As usual, by reading pa3.txt first….

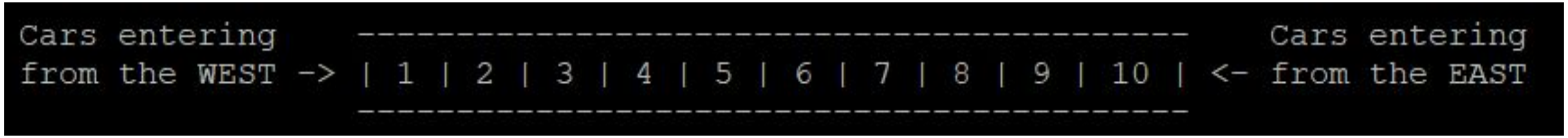
Summary of exercises

1. Pa3a.c : Get familiar with the road & driving system
2. Pa3b.c : Implement Semaphore and do a simple test
3. Pa3c.c : Get familiar with shared memory and do your experiments
4. Pa3d.c : Implement your traffic system! This file will be submitted as part of your solution.

First part PA3a : Understand the traffic system

Here you will be simulating traffic through a lane.

- Single lane, a single car takes up the whole width of the road.
- Each Car is a "process".
- The road:



```
Cars entering     ------------------------------------------     Cars entering
from the WEST -> | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | <- from the EAST
                  ------------------------------------------
```

- The cars (or processes) can enter from either the west or east side.
- The lane is 10 miles long and divided into 10 positions.

First part PA3a : Functions in driveRoad()

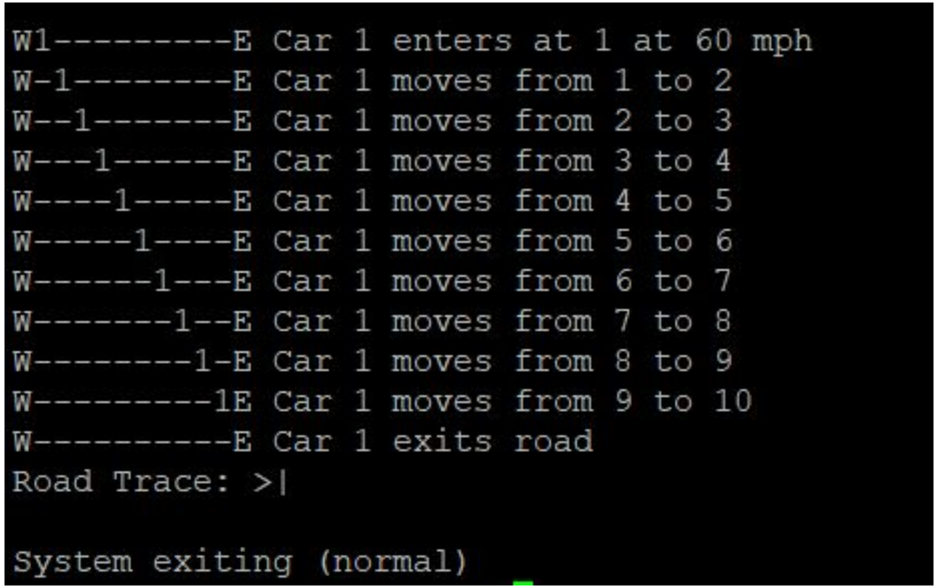
- printroad() function:
    - If we just call **PrintRoad()**, the output on the screen:
        - W----------E
    -
- EnterRoad (from):
    - Causes the can to enter the road from the direction mentioned in “from”.
    - If we do : **EnterRoad(EAST); PrintRoad()**; the output on screen:
        - W----------E
        - W---------1E
- ProceedRoad ():
    - Moves the car in the direction it Entered
    - Now if we do a **PrintRoad()**, the output on the screen:
        - W----------E
        - W---------1E
        - W--------1-E

Rules in this game:

- Each car has its own speed.
- Cars enter and proceed at a certain speed and maintain that speed until they exit.
    - So if you see a car coming at way high speed from behind, you have no option to increase your speed and escape.
- The primary rule of the road is that each position can be occupied by at most 1 car at any time. (No sharing of a position).
- Each car moves to next position based on its own speed.
    - Eg. Car A has a speed of 40 Mph and Car has a speed of 60 Mph.
    - Then every (3600/40) = 90 seconds, Car A will move to next position.
    - Every (3600/60) = 60 sec, Car B will move to next position.

Based on the rules we discussed, what happens when:

- Case 1: Car A enters from West at 60 Mph at time T= 0 sec. No other cars.
- Output:

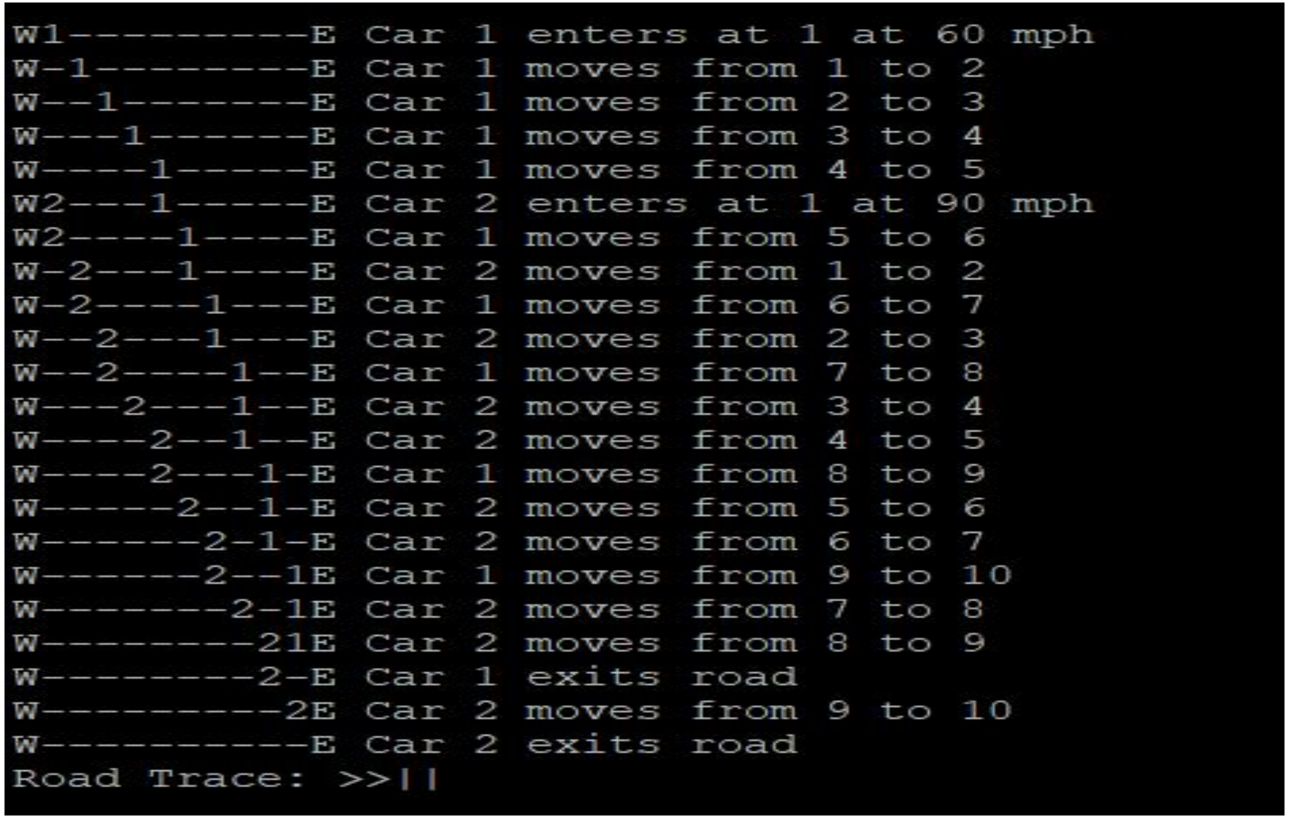


```
W1---------E Car 1 enters at 1 at 60 mph
W-1--------E Car 1 moves from 1 to 2
W--1-------E Car 1 moves from 2 to 3
W---1------E Car 1 moves from 3 to 4
W----1-----E Car 1 moves from 4 to 5
W-----1----E Car 1 moves from 5 to 6
W------1---E Car 1 moves from 6 to 7
W-------1--E Car 1 moves from 7 to 8
W--------1-E Car 1 moves from 8 to 9
W---------1E Car 1 moves from 9 to 10
W----------E Car 1 exits road
Road Trace: >|

System exiting (normal)
```

Based on the rules we discussed, what happens when:

- Case 2: Car A enters from West at 60 Mph at time T= 0 sec and Car B enters from West at 90 Mph at time T = 300 sec.
  - Car A takes 600 sec to exit
  - Car B takes 400 sec to exit

Case 2: Car A enters from West at 60 Mph at time T= 0 sec and Car B enters from West at 90 Mph at time T = 300 sec.



```
W1---------E Car 1 enters at 1 at 60 mph
W-1--------E Car 1 moves from 1 to 2
W--1-------E Car 1 moves from 2 to 3
W---1------E Car 1 moves from 3 to 4
W----1-----E Car 1 moves from 4 to 5
W2---1-----E Car 2 enters at 1 at 90 mph
W2----1----E Car 1 moves from 5 to 6
W-2---1----E Car 2 moves from 1 to 2
W-2----1---E Car 1 moves from 6 to 7
W--2---1---E Car 2 moves from 2 to 3
W--2----1--E Car 1 moves from 7 to 8
W---2---1--E Car 2 moves from 3 to 4
W----2--1--E Car 2 moves from 4 to 5
W----2---1-E Car 1 moves from 8 to 9
W-----2--1-E Car 2 moves from 5 to 6
W------2-1-E Car 2 moves from 6 to 7
W------2--1E Car 1 moves from 9 to 10
W-------2-1E Car 2 moves from 7 to 8
W--------21E Car 2 moves from 8 to 9
W--------2-E Car 1 exits road
W---------2E Car 2 moves from 9 to 10
W----------E Car 2 exits road
Road Trace: >>||
```

Based on the rules we discussed, what happens when:

- Case 3: Car A enters from East at 60 Mph at time T= 0 sec and Car B enters from **West** at 90 Mph at time T = 0 sec. At which position will they collide ?

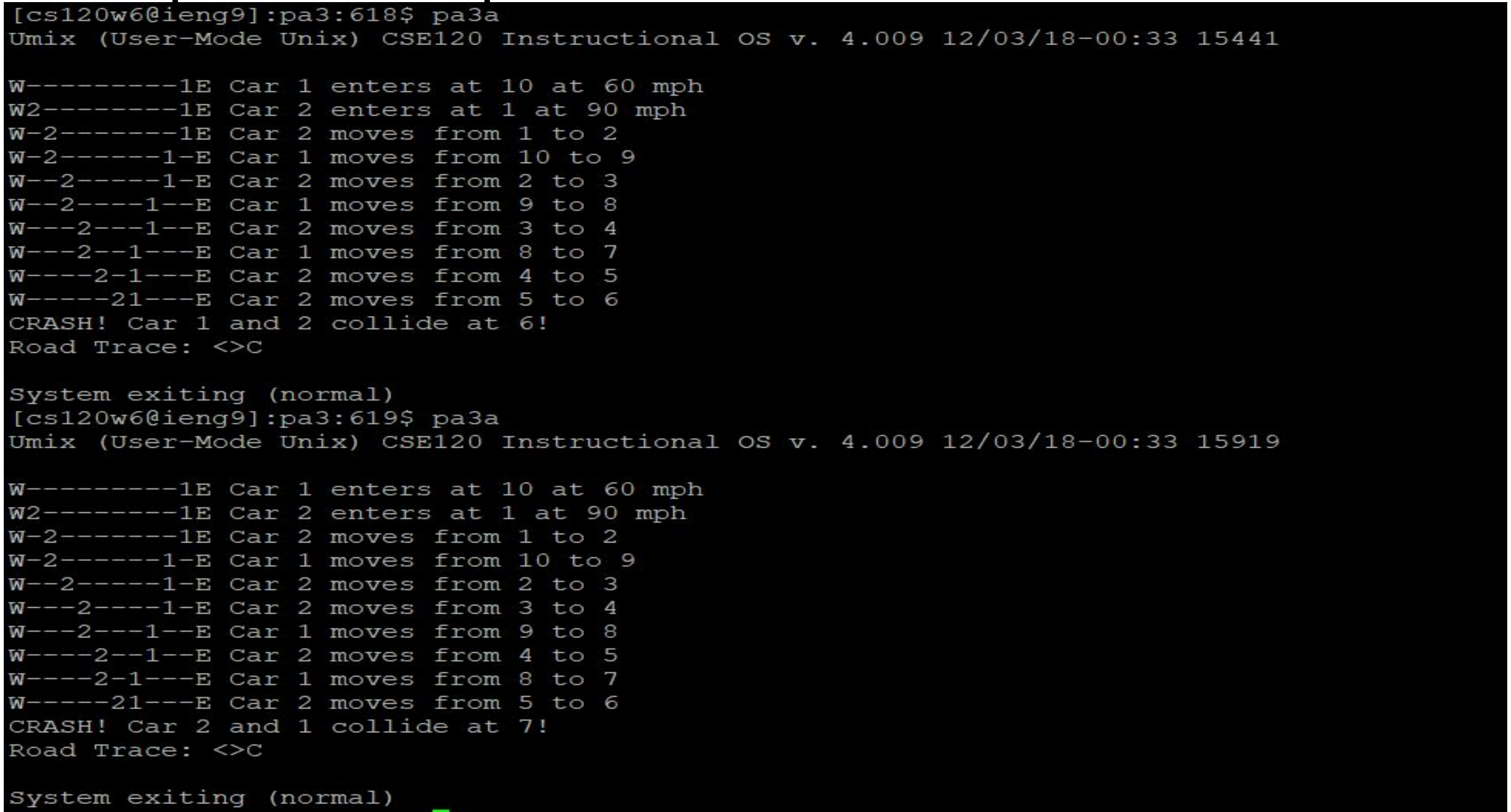
Based on the rules we discussed, what happens when:

- Case 3: Car A enters from East at 60 Mph at time T= 0 sec and Car B enters from **West** at 90 Mph at time T = 0 sec. At which position will they collide ?

Lets Simulate:

1. T = 0, Car A enters at position 10 and **then** Car B enters at position 1. **OR** Car B enters at position 1 and **then** Car A enters at position 10.
2. T = 40 sec: Car B at position 2.
3. T = 60 sec: Car A at position 9.
4. T = 80: Car B at position 3.
5. T = 120: Car B at position 4 and Car A at position 8.
6. T = 160: Car B from 4 to 5.
7. T = 180: Car A from 8 to 7.
8. T = 200: Car B from 5 to 6.
9. T = 240: Car A moves into position 6 & collides **OR** Car B moves into position 7 and collides.

Both possible in pa3a



```
[cs120w6@ieng9]:pa3:618$ pa3a
Umix (User-Mode Unix) CSE120 Instructional OS v. 4.009 12/03/18-00:33 15441

W---------1E Car 1 enters at 10 at 60 mph
W2--------1E Car 2 enters at 1 at 90 mph
W-2-------1E Car 2 moves from 1 to 2
W-2------1-E Car 1 moves from 10 to 9
W--2-----1-E Car 2 moves from 2 to 3
W--2----1--E Car 1 moves from 9 to 8
W---2---1--E Car 2 moves from 3 to 4
W---2--1---E Car 1 moves from 8 to 7
W----2-1---E Car 2 moves from 4 to 5
W-----21---E Car 2 moves from 5 to 6
CRASH! Car 1 and 2 collide at 6!
Road Trace: <>C

System exiting (normal)
[cs120w6@ieng9]:pa3:619$ pa3a
Umix (User-Mode Unix) CSE120 Instructional OS v. 4.009 12/03/18-00:33 15919

W---------1E Car 1 enters at 10 at 60 mph
W2--------1E Car 2 enters at 1 at 90 mph
W-2-------1E Car 2 moves from 1 to 2
W-2------1-E Car 1 moves from 10 to 9
W--2-----1-E Car 2 moves from 2 to 3
W---2----1-E Car 2 moves from 3 to 4
W---2---1--E Car 1 moves from 9 to 8
W----2--1--E Car 2 moves from 4 to 5
W----2-1---E Car 1 moves from 8 to 7
W-----21---E Car 2 moves from 5 to 6
CRASH! Car 2 and 1 collide at 7!
Road Trace: <>C

System exiting (normal)
```

Based on the rules we discussed, what happens when:

Case 4: Car A enters from West at 60 Mph at time T= 0 sec and Car B enters from West at 90 Mph at time T = 200 sec.

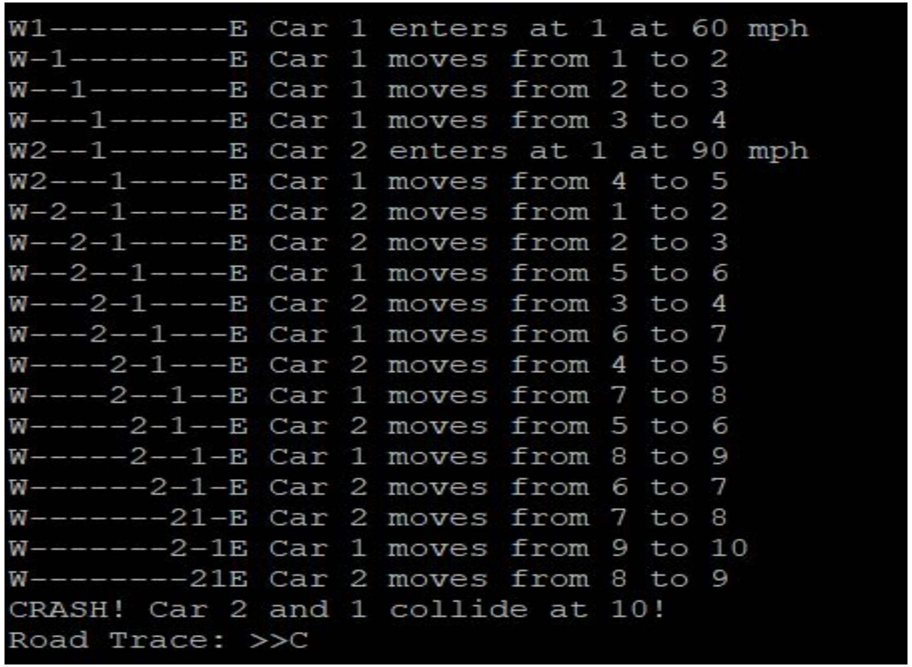
- Car A takes 600 sec to exit
- Car B takes 400 sec to exit

Case 4: Car A enters from West at 60 Mph at time T= 0 sec and Car B enters from West at 90 Mph at time T = 200 sec.

Will they Collide? If yes, where ?

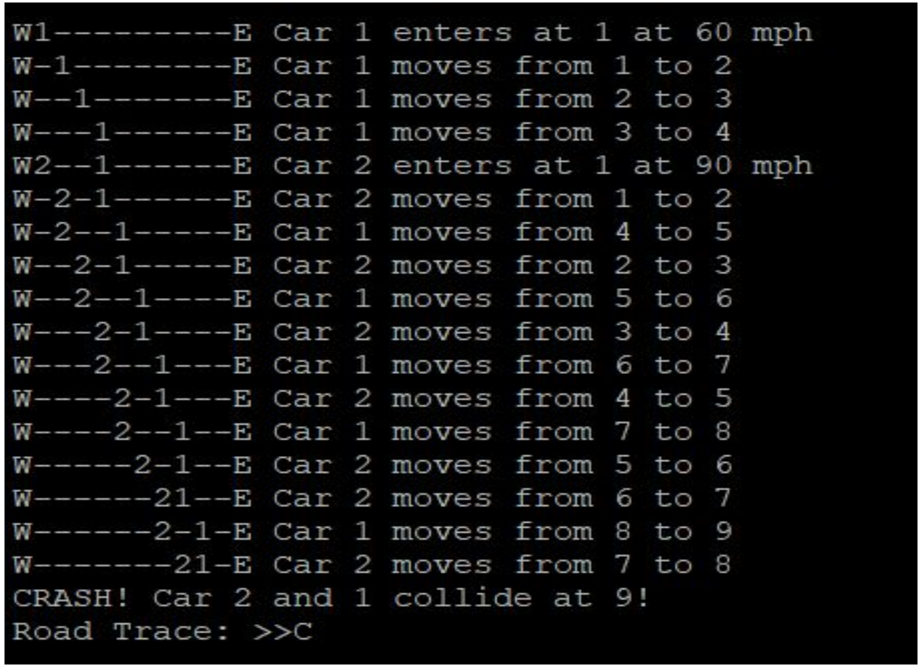
- Car A takes 600 sec to exit
- Car B takes 400 sec to exit

Run 1



```
W1---------E Car 1 enters at 1 at 60 mph
W-1--------E Car 1 moves from 1 to 2
W--1-------E Car 1 moves from 2 to 3
W---1------E Car 1 moves from 3 to 4
W2--1------E Car 2 enters at 1 at 90 mph
W2---1-----E Car 1 moves from 4 to 5
W-2--1-----E Car 2 moves from 1 to 2
W--2-1-----E Car 2 moves from 2 to 3
W--2--1----E Car 1 moves from 5 to 6
W---2-1----E Car 2 moves from 3 to 4
W---2--1---E Car 1 moves from 6 to 7
W----2-1---E Car 2 moves from 4 to 5
W----2--1--E Car 1 moves from 7 to 8
W-----2-1--E Car 2 moves from 5 to 6
W-----2--1-E Car 1 moves from 8 to 9
W------2-1-E Car 2 moves from 6 to 7
W-------21-E Car 2 moves from 7 to 8
W-------2-1E Car 1 moves from 9 to 10
W--------21E Car 2 moves from 8 to 9
CRASH! Car 2 and 1 collide at 10!
Road Trace: >>C
```

Run 2



```
W1---------E Car 1 enters at 1 at 60 mph
W-1--------E Car 1 moves from 1 to 2
W--1-------E Car 1 moves from 2 to 3
W---1------E Car 1 moves from 3 to 4
W2--1------E Car 2 enters at 1 at 90 mph
W-2-1------E Car 2 moves from 1 to 2
W-2--1-----E Car 1 moves from 4 to 5
W--2-1-----E Car 2 moves from 2 to 3
W--2--1----E Car 1 moves from 5 to 6
W---2-1----E Car 2 moves from 3 to 4
W---2--1---E Car 1 moves from 6 to 7
W----2-1---E Car 2 moves from 4 to 5
W----2--1--E Car 1 moves from 7 to 8
W-----2-1--E Car 2 moves from 5 to 6
W------21--E Car 2 moves from 6 to 7
W------2-1-E Car 1 moves from 8 to 9
W-------21-E Car 2 moves from 7 to 8
CRASH! Car 2 and 1 collide at 9!
Road Trace: >>C
```

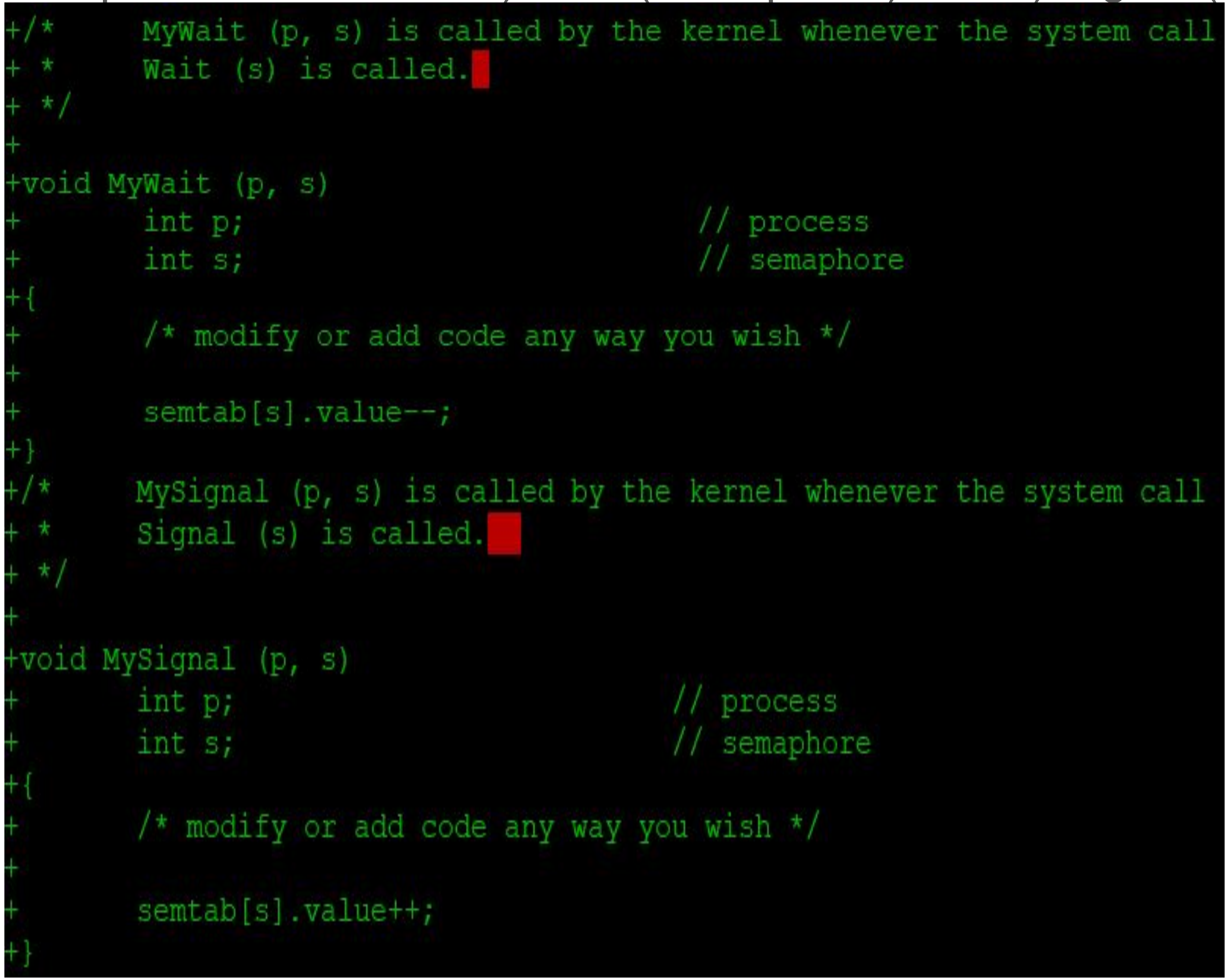
Why so much variations?

- Because of scheduler and Delay!
    - We can’t tell how much time the scheduler will take to context switch
    - Also the Delay(90) will not cause exact 90sec delay. There could be slight variations!

Part 2: Control traffic using semaphore

- Pa3b.txt already calls semaphore.
- Idea: Use semaphore to ensure that there is only one car on the lane at a time.
- But currently semaphore code is not complete and processes don’t wait on semaphore.

Complete the code for 1) Wait (semaphore) and 2) Signal (semaphore)



```
+/*     MyWait (p, s) is called by the kernel whenever the system call
+ *     Wait (s) is called.
+ */
+
+void MyWait (p, s)
+       int p;                          // process
+       int s;                          // semaphore
+{
+       /* modify or add code any way you wish */
+
+       semtab[s].value--;
+}
+/*     MySignal (p, s) is called by the kernel whenever the system call
+ *     Signal (s) is called.
+ */
+
+void MySignal (p, s)
+       int p;                          // process
+       int s;                          // semaphore
+{
+       /* modify or add code any way you wish */
+
+       semtab[s].value++;
+}
```

You are given two functions to assist you in this:

1) Block (p): causes process p to block. Removes p from being eligible for scheduling, and context switches to another process that is eligible.

2) Unblock(p): causes process p to be eligible for scheduling. The process p need not start executing immediately.

Semaphore wait and signal pseudocodes

Implementation 1

```
wait (sem s) {
    s.n = s.n - 1;
    if s.n < 0 {
        addProc (me, s.L);
        block (me);
    }
}

signal (sem s) {
    s.n = s.n + 1;
    if (! empty (s.L)) {
        p = removeProc (s.L);
        unblock (p);
    }
}
```

Implementation 2

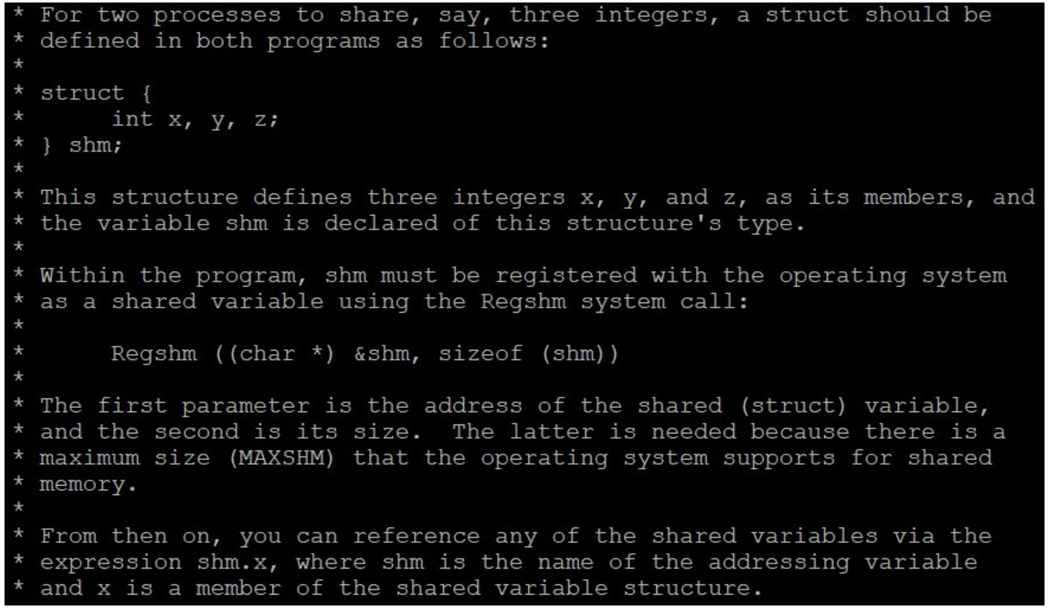
```
wait (sem s) {
    if s.n ≤ 0 {
        addProc (me, s.L);
        block (me);
    }
    s.n = s.n - 1;
}

signal (sem s) { // same
    s.n = s.n + 1;
    if (! empty (s.L)) {
        p = removeProc (s.L);
        unblock (p);
    }
}
```

Part 2: Control traffic using semaphore

Note: Semaphores are global resources. Once a semaphore is initialized by a process, it is accessible by all user processes.

Why : Semaphores are allocated in kernel and user processes initialize and access semaphores by passing an integer number to the kernel. Processes are not allocating memory for the semaphores they use.

Part 3: Using shared memory

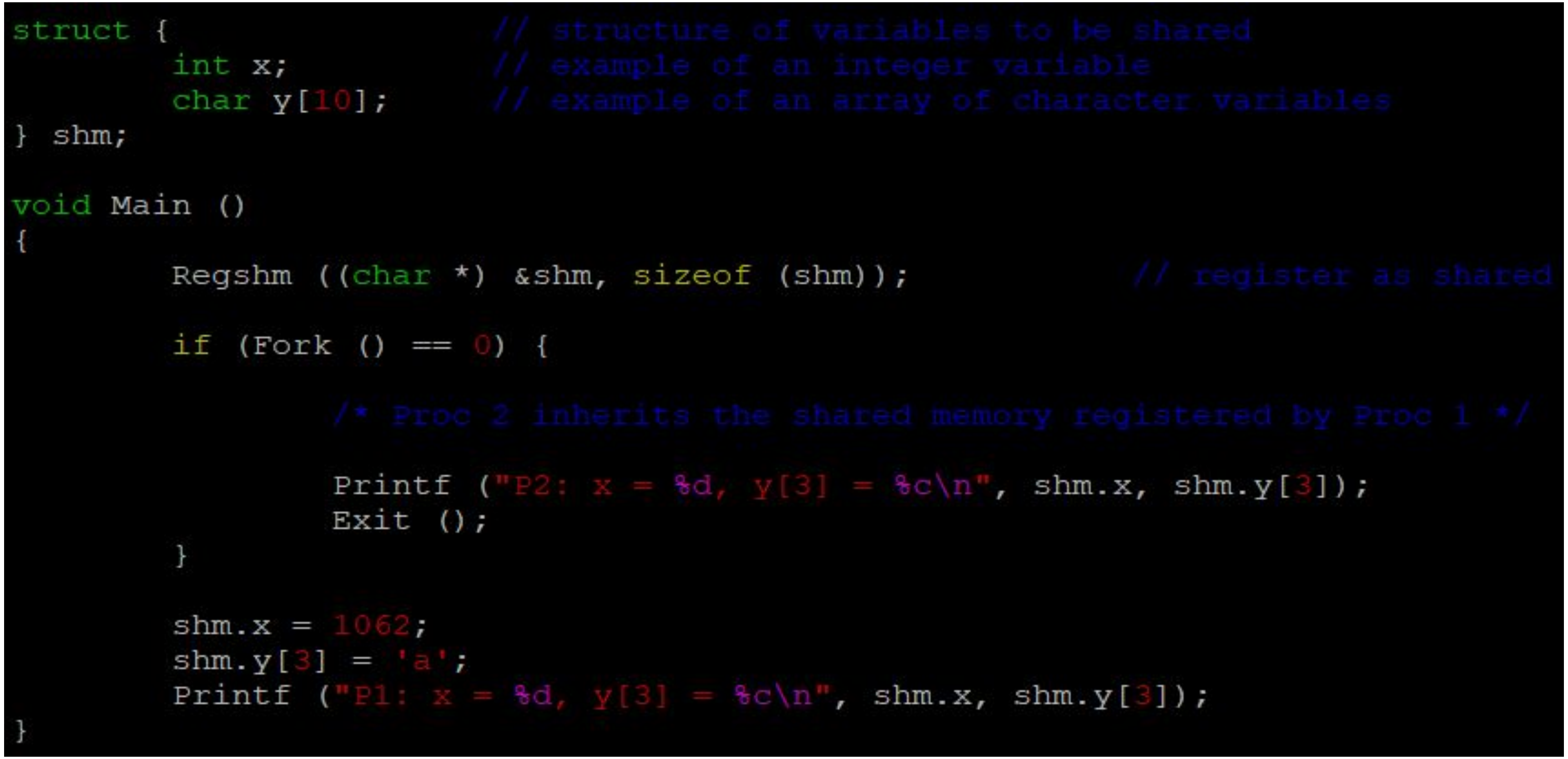


```
* For two processes to share, say, three integers, a struct should be
* defined in both programs as follows:
*
* struct {
*      int x, y, z;
* } shm;
*
* This structure defines three integers x, y, and z, as its members, and
* the variable shm is declared of this structure's type.
*
* Within the program, shm must be registered with the operating system
* as a shared variable using the Regshm system call:
*
*      Regshm ((char *) &shm, sizeof (shm))
*
* The first parameter is the address of the shared (struct) variable,
* and the second is its size.  The latter is needed because there is a
* maximum size (MAXSHM) that the operating system supports for shared
* memory.
*
* From then on, you can reference any of the shared variables via the
* expression shm.x, where shm is the name of the addressing variable
* and x is a member of the shared variable structure.
```

To make a variable shared between processes, use Regshm() command.

Variables that become shared memory should not be used prior to the call. If they are used, after calling Regshm(<variable name>), whatever value was stored prior to Regshm call will get wiped out

Part 3: Using shared memory: The code



```
struct {                        // structure of variables to be shared
        int x;                  // example of an integer variable
        char y[10];             // example of an array of character variables
} shm;

void Main ()
{
        Regshm ((char *) &shm, sizeof (shm));           // register as shared

        if (Fork () == 0) {

                /* Proc 2 inherits the shared memory registered by Proc 1 */

                Printf ("P2: x = %d, y[3] = %c\n", shm.x, shm.y[3]);
                Exit ();
        }

        shm.x = 1062;
        shm.y[3] = 'a';
        Printf ("P1: x = %d, y[3] = %c\n", shm.x, shm.y[3]);
}
```

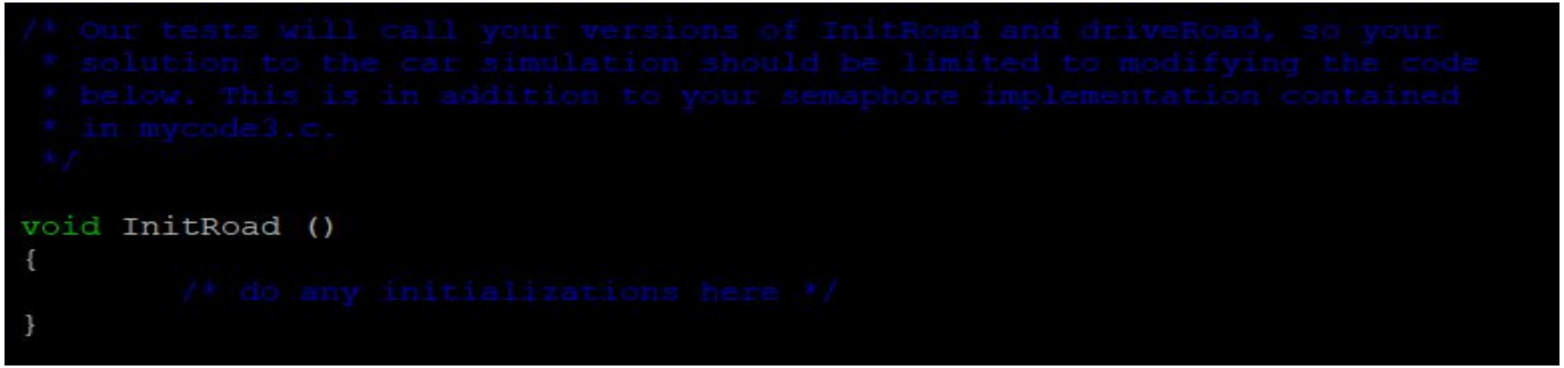
Part 4: The main part :)

Implement the driveRoad function in pa3d.c.

- Any process which calls driveRoad should never crash and finally its car should exit the road.
- You need to use semaphores and shared memory to achieve this.

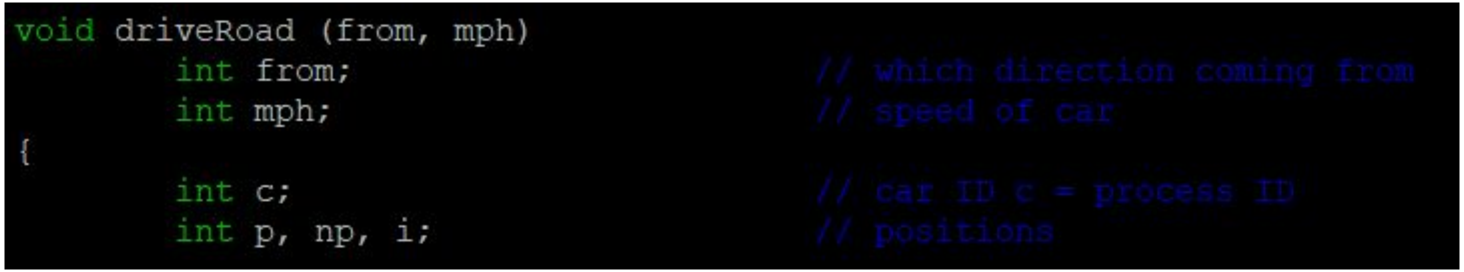
The functions needing implementation in pa3d

InitRoad() - > similar to InitSched() in PA2



```
/* Our tests will call your versions of InitRoad and driveRoad, so your
 * solution to the car simulation should be limited to modifying the code
 * below. This is in addition to your semaphore implementation contained
 * in mycode3.c.
 */

void InitRoad ()
{
        /* do any initializations here */
}
```

driveRoad() -> Similar to SchedProc() in PA2



```
void driveRoad (from, mph)
        int from;                    // which direction coming from
        int mph;                     // speed of car
{
        int c;                       // car ID c = process ID
        int p, np, i;                // positions
```

Some functions we will use in driveRoad() to simulate cars on the Road

- EnterRoad(from) : Makes the car enter the road
- ProceedRoad(): Moves the car

1) Modify the procedure driveRoad such that the following rules are obeyed:

A) Avoid all collisions.

B) Multiple cars should be allowed on the road, as long as they are traveling in the same direction.

C) If a car arrives and there are already other cars traveling in the SAME DIRECTION, the arriving car should be allowed enter as soon as it can. Two situations might prevent this car from entering immediately: (1) there is a car immediately in front of it (going in the same direction), and if it enters it will crash (which would break rule A); (2) one or more cars have arrived at the other end to travel in the opposite direction and are waiting for the current cars on the road to exit, which is covered by the next rule.

D) If a car arrives and there are already other cars traveling in the OPPOSITE DIRECTION, the arriving car must wait until all these other cars complete their course over the road and exit. It should only wait for the cars already on the road to exit; no new cars traveling in the same direction as the existing ones should be allowed to enter.

E) The previous rule implies that if there are multiple cars at each end waiting to enter the road, each side will take turns in allowing one car to enter and exit. However, if there are no cars waiting at one end, then as cars arrive at the other end, they should be allowed to enter the road immediately.

F) If the road is free (no cars), then any car attempting to enter should not be prevented from doing so.

G) All starvation must be avoided. For example, any car that is waiting must eventually be allowed to proceed.

Guidelines

Can only add synchronization and use shared memory, cannot decide delay, speed or direction of cars.

IMP: You will be penalized if your semaphores are not correctly implemented, even if your driveRoad() works.

When using shared memory, avoid race conditions. (use semaphore!)

What to turn in ?

1. In mycode3.c :
   a. implementation of semaphores
2. In pa3d.c :
   a. your modified version of InitRoad and driveRoad (Main will be ignored).
   b. You can add new functions and static shared memory in pa3d.c and use them within InitRoad & driveRoad functions. The testing code will only call your InitRoad and driveRoad functions.

How your code will be tested

- Your semaphore implementation will be tested directly (without using any cars) for its correctness.
- Then your implementation of InitRoad and driveRoad functions will be tested. For this, a correct semaphore implementation in the test code will be used (so that, even if your semaphore implementation is wrong, you can still get full credit on the driving part of the grade).

The semaphore in the testing code will unblock processes in FIFO order