CSE 120
PA3 Discussion – Part 1

PA3

- Main objective: Synchronizing bi-directional traffic on a single lane using semaphores

- You will implement wait() and signal() functions used by semaphores, and use semaphores in a user program function designed by you, to follow all the rules mentioned in the write-up

- mykernel3.c – Your semaphore implementation
- pa3d.c – Your user program

Part A

W | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10| E

- **Road** – Road is 10 miles long and consists of 10 positions (1 mile for each position).
  Cars may enter the road from west (position 1) or east (position 10). Cars enter the road with a certain speed and keep their speed while driving.

- **driveRoad ()** – called by cars who want to drive on the road.

- Understand the entire given code within driveRoad (), since you would again see this in part d.

- Each position can be occupied by at most 1 car at a time. If a car tries to proceed to a position that is already occupied, **Crash!!!**
  Examples of crash:
  1. Chase : Cars travelling in the same direction eventually collide if the 2nd car is much faster than the 1st car.
  2. Cars travelling in the opposite direction collide with each other.

- Intuition: Each position can be thought of as a resource guarded by a semaphore.

Part B

- Implementation of Wait() and Signal()

pa3*.c	Mykernel3.c
int s = Seminit (int v)	Int s = MySeminit (int p, int v)
Wait (int s)	MyWait (int p, int s)
Signal (int s)	MySignal (int p, int s)

MySeminit:
- Find an available semaphore
- Initialize value to v

MyWait:
- Decrement sem value
- If < 0 :
  put into waiting queue;
  Block(p);

MySignal:
- Increment sem value
- Unblock() if there's any process waiting

More about Part B

- Make sure you have a perfect implementation of Part B before proceeding. Test your code !!

- Common issues
  1. Make sure you do not have any code after Block() call. This code would not run since the process is blocked.
  2. Make sure that blocked processes are correctly being added to the semaphore queue.
  3. Avoid starvation. If multiple processes are blocked on the same semaphore, then a good policy is to unblock them in FIFO order.

Part C

- Sharing memory among processes (cars)
- We never did this in PA1 or PA2. So why is this needed in PA3?
- Put all the variables you want to share in a single struct, and use the Regshm() system call **exactly once**.

```
struct {
   int x, y, z;
} shm;

Regshm ((char *)&shm,  sizeof (shm));
```

- If a process has registered for shared memory, all the children of that process have access to that shared memory region. (need not call Regshm() again)

Part D

- For a change, you will be adding code to the user program, and not the kernel !

- **InitRoad()** – Add code here to initialize any semaphores you might want to use in part d.

- This is a good place to share variables (using Regshm)

Part D

- **driveRoad()** – This is the toughest part, and also the most important part of your implementation!!
- Here is where you would use semaphores to synchronize movements of multiple cars on the single lane.

Part D – Rules to follow

1. Avoid all collisions.

2. Multiple cars should be allowed on the road, as long as they are travelling in the same direction.

3. If the previous car is at the first position, then the next car in the same direction should not be allowed to enter since that would lead to a crash.

4. If a car arrives and there are already other cars travelling in the OPPOSITE direction, then the arriving car must wait until all these running cars complete their course over the road and exit.

   Important Note: It should only wait for the cars already on the road to exit. No new cars travelling in the same direction as the existing ones should be allowed to enter.

5. If there are multiple cars at each end waiting to enter the road, then each side should take turns to in allowing one car to enter and exit.

Part D – Rules to follow contd…

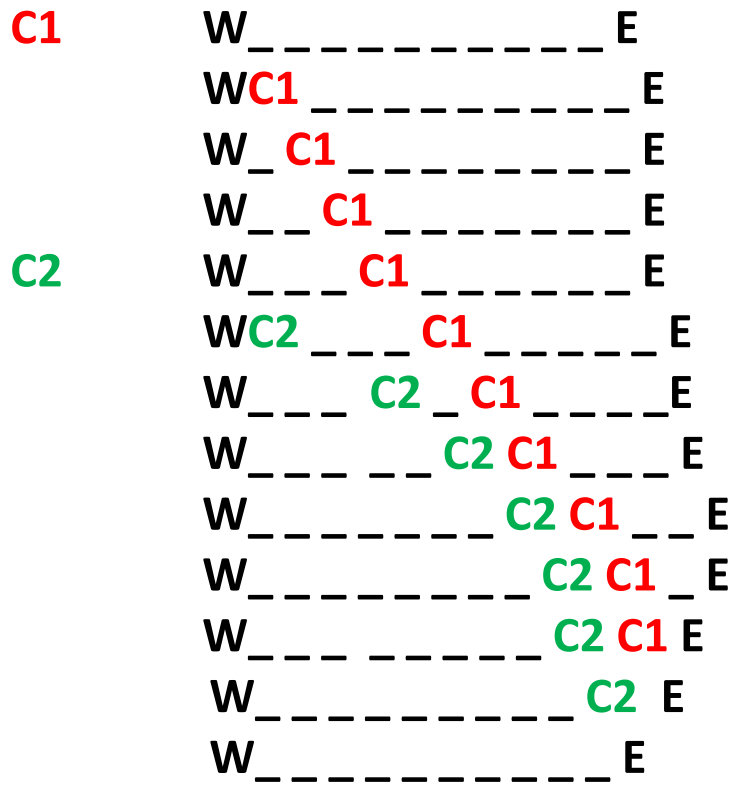
6. If there are no cars waiting at one end, then as cars arrive at the other end, they should be allowed to enter the road immediately.

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

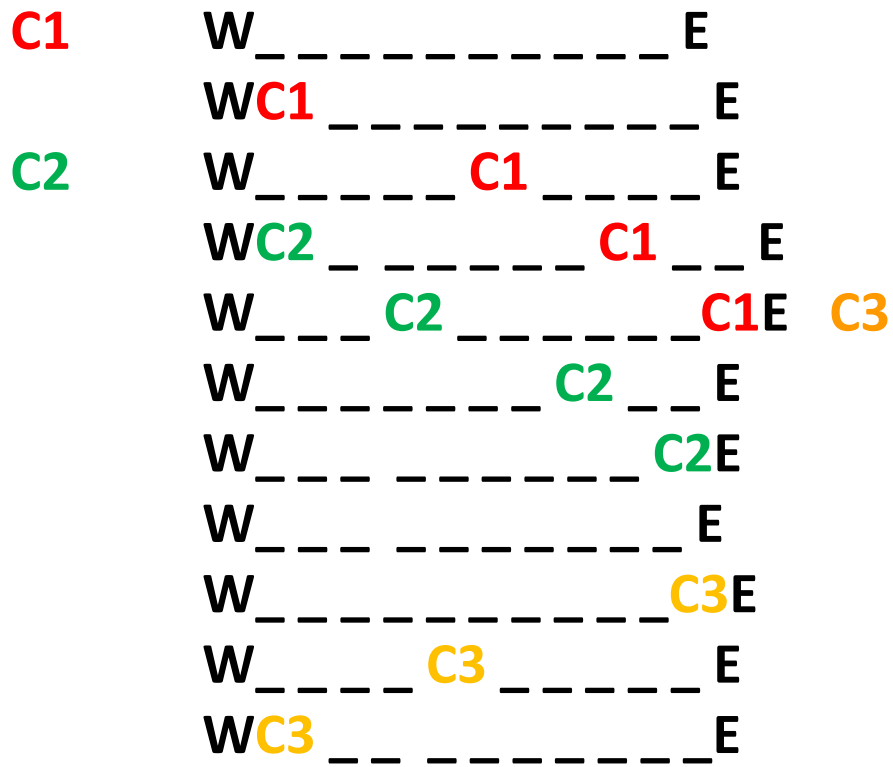
8. All starvation must be avoided .i.e. a car must have a chance to enter the road sooner or later.

9. Two parameters you DO NOT have control over:
   - The time and order of arrival of cars.
   - The speed of the cars

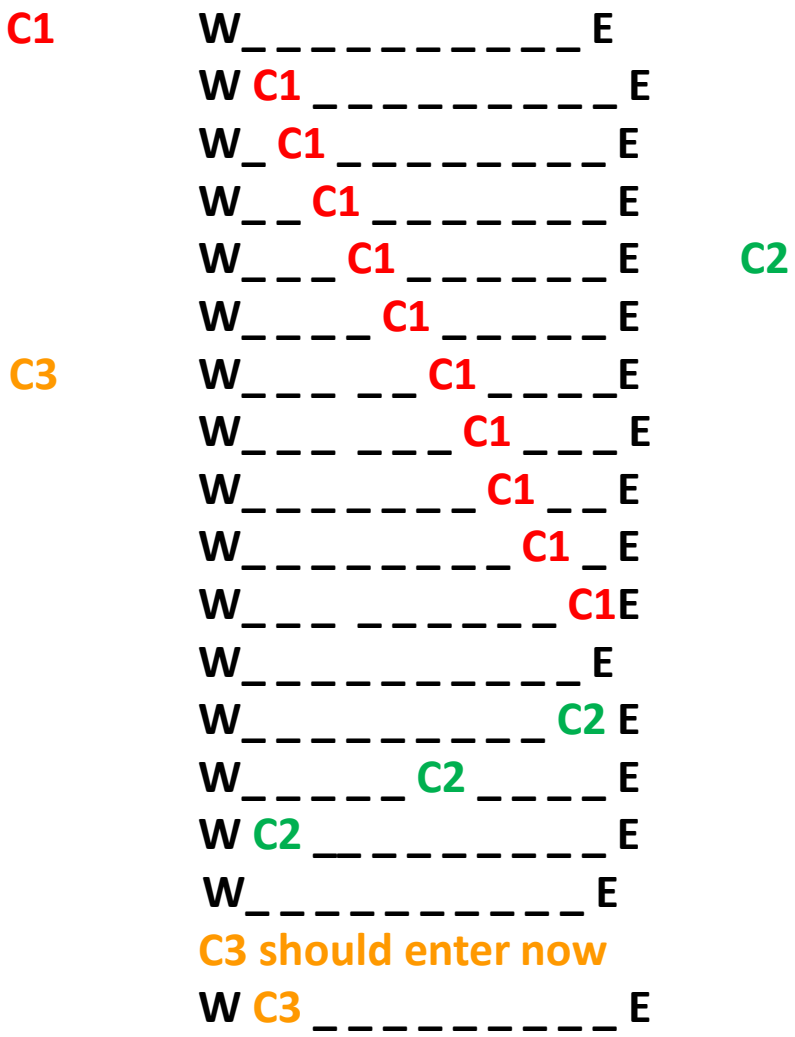
A few practice runs..
Level 0 – C2 is much faster than C1



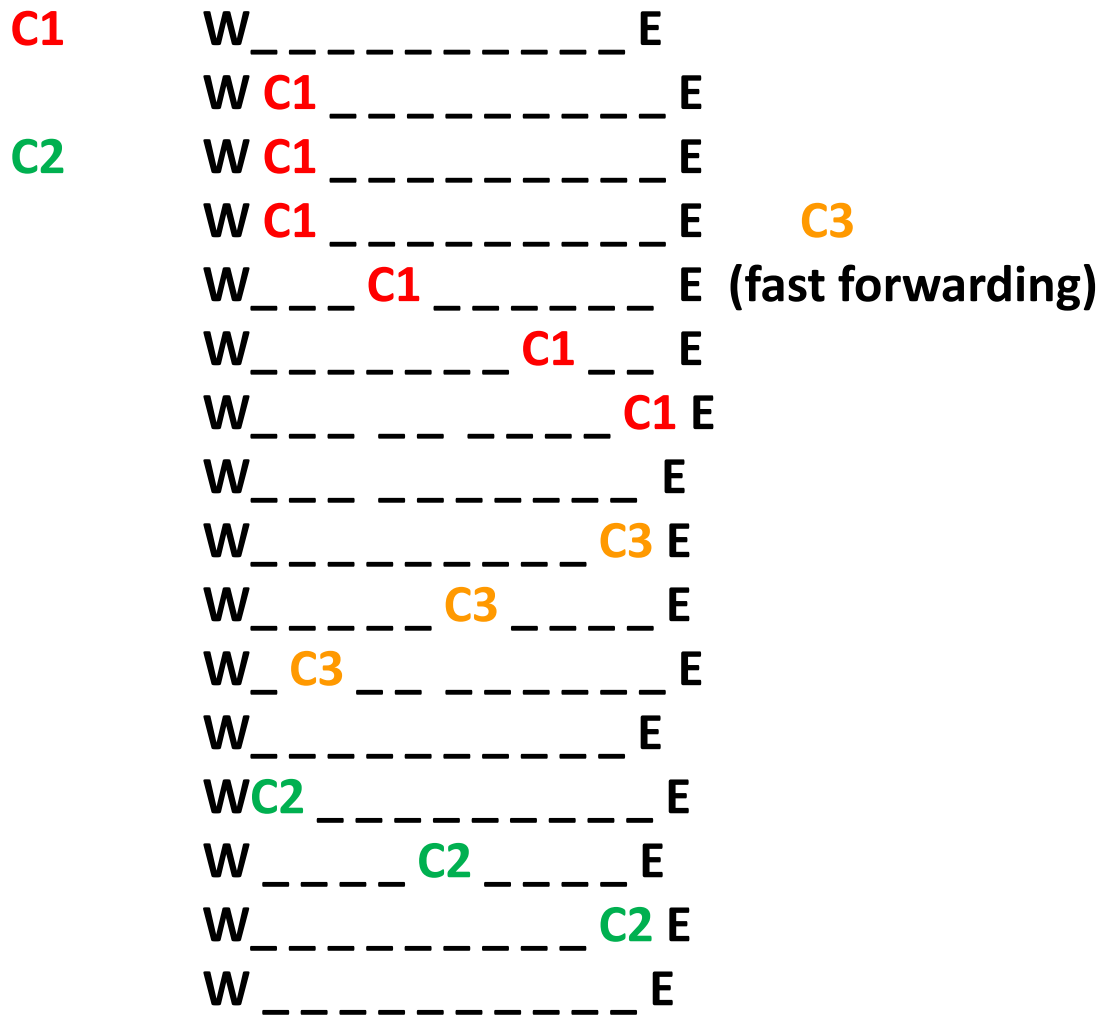
Level 1


C1
W_ _ _ _ _ _ _ _ _ _ E
WC1 _ _ _ _ _ _ _ _ _ E
C2
W_ _ _ _ _ C1 _ _ _ _ E
WC2 _ _ _ _ _ _ C1 _ _ E
W_ _ _ C2 _ _ _ _ _ _C1E C3
W_ _ _ _ _ _ _ C2 _ _ E
W_ _ _ _ _ _ _ _ C2E
W_ _ _ _ _ _ _ _ _ E
W_ _ _ _ _ _ _ _ _ _C3E
W_ _ _ _ C3 _ _ _ _ _ E
WC3 _ _ _ _ _ _ _ _ _E

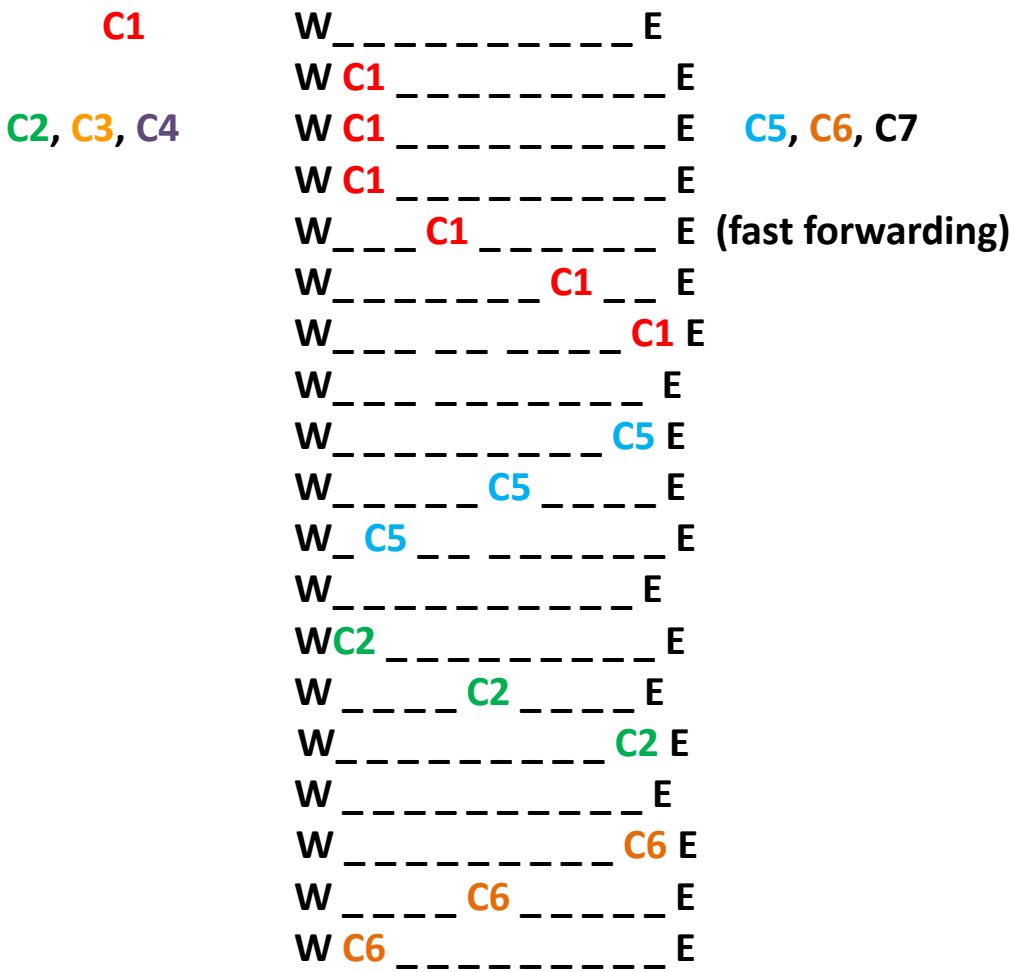
Level 2


C1
W_ _ _ _ _ _ _ _ _ _ E
W C1 _ _ _ _ _ _ _ _ E
W_ C1 _ _ _ _ _ _ _ E
W_ _ C1 _ _ _ _ _ _ E
W_ _ _ C1 _ _ _ _ _ E
C2
W_ _ _ _ C1 _ _ _ _ E
C3
W_ _ _ _ _ C1 _ _ _ E
W_ _ _ _ _ _ C1 _ _ E
W_ _ _ _ _ _ _ C1 _ E
W_ _ _ _ _ _ _ _ C1E
W_ _ _ _ _ _ _ _ _ _ E
W_ _ _ _ _ _ _ _ _ C2 E
W_ _ _ _ _ C2 _ _ _ _ E
W C2 _ _ _ _ _ _ _ _ E
W_ _ _ _ _ _ _ _ _ _ E
C3 should enter now
W C3 _ _ _ _ _ _ _ _ E

Level 3
Extremely slow first car



Level 4
Extending this..



Basic idea is you need to allow 1 car from each direction

System back to initial state after all cars exit…

C1 W_ _ _ _ _ _ _ _ _ _ E

W C1 _ _ _ _ _ _ _ _ _ E

C3 W C1 _ _ _ _ _ _ _ _ E C2

W C1 _ _ _ _ _ _ _ _ E

W_ _ _ C1 _ _ _ _ _ _ E (fast forwarding)

W_ _ _ _ _ _ _ C1 _ _ E

W_ _ _ _ _ _ _ _ _ C1 E

W_ _ _ _ _ _ _ _ _ _ E

W_ _ _ _ _ _ _ _ _ C2 E

W_ _ _ _ _ C2 _ _ _ _ E

W_ C2 _ _ _ _ _ _ _ _ E

W_ _ _ _ _ _ _ _ _ _ E

WC3 _ _ _ _ _ _ _ _ _ E

W _ _ _ _ C3 _ _ _ _ E

W_ _ _ _ _ _ _ _ _ C3 E

W _ _ _ _ _ _ _ _ _ _ E (the initial set of cars have all left the road)

W _ _ _ _ _ _ _ _ _ _ E

W _ _ _ _ _ _ _ _ _ _ E C4

W _ _ _ _ _ _ _ _ _ C4 E

W C4 _ _ _ _ _ _ _ _ _ E

Hints

- Define a semaphore for each position of the road so that only one car could occupy a position because a CRASH will occur if a car tries to proceed to a position that is already occupied.
  - **int road[NUMPOS]**

Hints

- Say a new car **C** arrives and the first position is available. Can it enter the road always? No. You cannot always enter the road if the first position is available.

  Why?

  Case1: There might be some cars travelling on the road in the opposite direction. So, in this case, you need a semaphore say **'door'** (one for each direction) to prevent car C from entering.

  Case2: There might be some cars waiting to enter the road in the opposite direction.

  As per the rules, a car should wait only for the cars which were already on the road when it arrived and not for this new car C. Hence, you will need a semaphore say **'ticket'** to prevent car C from entering the road, if there are cars waiting to enter the road from the opposite direction.

Hints

- A car should first wait for the “first position to become available” even before it waits for the ‘ticket’ and ‘door’ semaphores.

  Therefore, the first position will need another semaphore say **‘queue’** (one for each direction) in addition to the “road” semaphore. So the first position will have two semaphores.

Few more things to keep in mind

- Extend your test cases to lot more than 4 cars. Your implementation should work with MAXPROCS cars.
- Try scenarios where in real life you would definitely get a crash.
- Examples:
  - 2nd car much faster than 1st car in same direction – good test case
  - 1st car much faster than 2nd car in the same direction – may not be a good test case

Analogy with Readers-Writers Problem

- 1st Readers Writers problem states: No reader should wait for another reader to finish his task < == > Cars in the same direction should be allowed to enter the track.

- 2nd Readers Writers problem states: If a writer is waiting to access a object, no new readers may start reading < == > Once a car from opposite direction arrives, no new cars from the current direction should be allowed to pass.

- But both these problems suffer from starvation drawback.

- You need a solution which avoids these drawbacks. Check out stuff on the Readers-Writers problem in the text book / internet.