1.

```
monitor network{
    int sum = 0;
    condition c;
    Semaphore Mutex = 1;

    int init(n){
        return n;
    }
    void request(int n){
        wait(Mutex);
        while(sum + n > n)
            c.wait();
        sum += n;
        signal(Mutex);
    }
    void release(int n){
        wiat(Mutex);
        sum += n;
        c.broadcase();
        signal(Mutex);
    }
    main(){
        requesr(init(n));
        release(init(n));
    }
}
```

2. IDK

3.

```
void init(int n) {
    Semaphore Mutex = 1;
    Semaphore LeftSeats = n;
    Semaphore Customers = 0;
    Semaphore Chef = 0;
}
Chef(){
    wait(Customers);
    wiat(Mutex);
    LeftSeats++;
    signal(Chef);
    signal(Mutex);
}
Customer(){
    wait(Mutex);
    if(LeftSeats > 0){
        LeftSeats--;
        signal(Chef);
        signal(Mutex);
        wait(Chef);
    }else
        signal(Mutex);
}
```

4.a)

1. Mutual exclusion: only one car could occupy each space on the roadway.
2. Hold and wait: the car can stay and wait before the intersection till it is available to the car.
3. No preemption: the car cannot be removed from the intersection.
4. Circular wait: all cars form a cycle; each car is wait for next car, mand cars later, it will go back to the same car.

4.b)

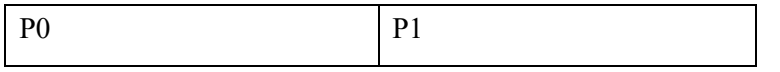
Some traffic lights might help to avoid the deadlocks. Each green light lets cars pass 45 mins, and make sure the horizontal and vertical traffic lights won't be same, such as both green/red won't happen.

5.

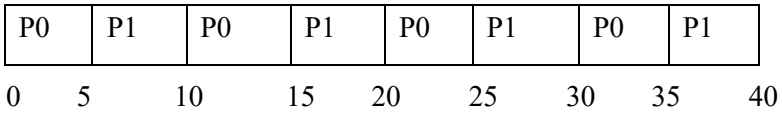
Deadlock is impossible. We can assume there is a deadlock in this system. The condition that causes 2 threads to wait would be that each thread has the resources, and one thread has 2 resources. If there is a thread that has two recourses, we only need to wait for this thread to finish and release it. It's contradictory with our assumption.

6.

FCFS



RR



we can see that Round Robin will lead to a deadlock.

7.

a) No deadlock. T2 -> T3 -> T1

b) has a deadlock. T1->R3->T3->R1->T1

T1 holds an instance of R1 and is waiting for an instance of R3

T3 holds an instance of R3 and is waiting for an instance of R1

c) No deadlock. T2->T3->T1

d) has a deadlock. R1->T1, T2->R2, T4->R1

T1 holds one instance of R1 and is waiting for one instance of R2

T2 holds one instance of R1 and is waiting for one instance of R2

T3 holds one instance of R2 and is waiting for one instance of R1

T4 holds one instance of R2 and is waiting for one instance of R1

e) No deadlock. T2->T1->T4->T3

f) No deadlock. T4->T2->T1->T3