
Fast Nuces selects a student to solve teen talwar traffic intersection problem.The details of problem is as follows: in two one-way roads as shown above, traffic is trying to pass through this intersection of roads, such as east to west and north to south without any collision,the intersection road is origin point for cars. Cars have to exit in the opposite direction, therefore cars need to pass through intersections. In this problem, cars are not allowed to take turns. Inside an intersection, more than one vehicle can get in and given that a car's path will not create a collision. Due to some construction problems in this area, a maximum of three cars can get inside the intersection at the same time. Create a program with proper synchronization as per provided scenario.

**Solution:**

```
// Define your global variables here.
#define MAX_CARS_IN_INTERSECTION 3
enum Directions {
north = 0, east = 1
};

typedef enum Directions Direction;

// Volatile is not necessary because of the use of locks and condition variables
// to provide mutual exclusion and synchronization.
struct lock* l;
struct cv* direction_cv[2];
int num_waiting[2];
int num_inside;
```

```
Direction inter_direction; // Direction inside the intersection.
Direction opposing_direction(Direction d) {
return (d == north) ? east : north;
}

// Called only once, before any vehicles try to enter the intersection.
void intersection_sync_init(void) {
l = lock_create("lock");
direction_cv[north] = cv_create("North/South CV");
direction_cv[east] = cv_create("East/West CV");
num_waiting[north] = 0;
num_waiting[east] = 0;
num_inside = 0;
inter_direction = north; // Arbitrary starting direction.
}

// Called only once, at the end of the simulation.
void intersection_sync_cleanup(void) {
cv_destroy(direction_cv[north]);
cv_destroy(direction_cv[east]);
direction_cv[north] = NULL;
direction_cv[east] = NULL;
lock_destroy(l);
l = NULL;
}

// Called by a vehicle simulation before a vehicle enters the intersection.
void intersection_before_entry(Direction origin) {
lock_acquire(l);

// Car is waiting (at least momentarily) at the intersection entrance.
num_waiting[origin]++;

// Check if there is at least one car waiting in the opposing direction. If
// there is, then this car, which is just arriving at the intersection,
// should wait in order to provide temporal ordering. Note that a while
// loop is not necessary for this cv_wait since it is only used to modify
// the ordering of cars, and does not affect the safety of the
// intersection.

if (num_waiting[opposing_direction(origin)] > 0) {
cv_wait(direction_cv[origin], l);
} else if (num_inside == 0) {
```

```
// Intersection is empty, and no one is waiting in the opposing
// direction. Change the direction of the intersection to allow
// this car to (possibly) go inside the intersection.

inter_direction = origin;
}

// Wait if the number of cars inside the intersection >= the
// max number of cars. Also wait if the direction inside the
// intersection is opposing this car's direction. Note that
// barging is possible with this solution, but it is unlikely
// to have a large effect on fairness in practice.

while (num_inside >= MAX_CARS_IN_INTERSECTION || inter_direction != origin) {
cv_wait(direction_cv[origin], l);
}
num_waiting[origin]--;
num_inside++;
lock_release(l);
}
// Called by a vehicle simulation after a vehicle leaves the intersection.

void intersection_after_exit(Direction origin) {
lock_acquire(l);
num_inside--;

// Determining the opposing direction.

Direction od = opposing_direction(origin);
if (num_inside == 0) {
// If the intersection is empty, and there are cars waiting
// in the opposing direction, change the direction of traffic.
if (num_waiting[od] > 0)
inter_direction = od;
}
// Broadcast to wake up the selected direction. A more efficient
// solution would be to just call cv_signal
// min(num_waiting, MAX_CARS_IN_INTERSECTION) number of times.
cv_broadcast(direction_cv[inter_direction], l);
} else if (num_waiting[od] == 0 && num_waiting[origin] > 0) {
// No cars are waiting in the opposing direction. Let another
// car in the same direction go if there is one waiting.
cv_signal(direction_cv[origin], l);
}
```

```
lock_release(l);
}
```