Operating Systems
Documentation

No more than 3 cars travel

The cars on the road are calculated by

a) Incrementing by 1 every time a car enters a road
b) Subtracting by 1 every time a car leaves the road

This is achieved by using the entering and leave functions. The entering function would wait using a *conditional variable* if three cars were on the road.

Traveling cars at any time have the same direction

The incoming_onstreet and outgoing_onstreet cars keep records of cars on the street either incoming or outgoing. These variables were checked on both entering functions so that if any thread was traveling in the opposite direction the car would simply wait till the car was free.

This conditional variable would receive a signal from the street thread since it had a continuous loop and would alert the thread each time the issue would be resolved.

```
            pthread_mutex_unlock(&locks);
        }*/
        if (cars_since_repair==7 && cars_on_street==0){
            pthread_mutex_lock(&lock);
            repair_street();
            pthread_cond_signal(&condition_repairs);
            cars_since_repair = 0;
            pthread_mutex_unlock(&lock);
        }

        if (incoming_onstreet==0 && outgoing_onstreet<3){
            pthread_mutex_lock(&lock);
            pthread_cond_signal(&condition_outgoing_enter);
            pthread_mutex_unlock(&lock);
        }

        if (incoming_onstreet<3 && outgoing_onstreet==0){
            pthread_mutex_lock(&lock);
            pthread_cond_signal(&condition_incoming_enter);
            pthread_mutex_unlock(&lock);
        }

        if (cars_on_street<3){
            pthread_mutex_lock(&lock);
            pthread_cond_signal(&condition_three);
            pthread_mutex_unlock(&lock);
        }


        /*repair_street();*/


    }

    pthread_exit(NULL);
}
```

The street repairs after every 7th car

Every time a car exits the street we increment the cars since the repair variable by keeping track of the cars that travel the street since the repair. This number is then toggled back to zero every time the street is repaired. The threads wait till the streets are empty before starting the repair.

This is achieved in the final approach by waiting in the incoming and outgoing threads before entering. This approach allows for the threads to take a break when conditions are not being met.

Example of one such thread is given below:

```
void*
incoming_thread(void *arg) {
    car *car_info = (car*)arg;

    pthread_mutex_lock(&locktry);

    if (cars_on_street>=3){
        pthread_cond_wait(&condition_three,&locktry);
    }
    if (outgoing_onstreet>0){
        pthread_cond_wait(&condition_incoming_enter,&locktry);
    }
    if (cars_since_repair>=7){
        pthread_cond_wait(&condition_repairs,&locktry);
    }
    /* enter street */
    incoming_enter();


        /* Car travel --- do not make changes to the 3 lines below*/
    printf("Incoming car %d has entered and travels for %d minutes\n", car_info->car_id, car_info->travel_
    travel(car_info->travel_time);
    printf("Incoming car %d has travelled and prepares to leave\n", car_info->car_id);

    /* leave street */

    incoming_leave();
    pthread_mutex_unlock(&locktry);

    pthread_exit(NULL);
}
```

We achieved maximum concurrency by waiting each time conditions were not fulfilled and allowing for multiple threads to coexist without violating any of the rules. We achieved this by using one main lock and four conditional variables:

```
pthread_cond_t condition_incoming_enter = PTHREAD_COND_INITIALIZER;
pthread_cond_t condition_outgoing_enter = PTHREAD_COND_INITIALIZER;
pthread_cond_t condition_three = PTHREAD_COND_INITIALIZER;
pthread_cond_t condition_repairs = PTHREAD_COND_INITIALIZER;
```