Programming Project Checkpoint 5

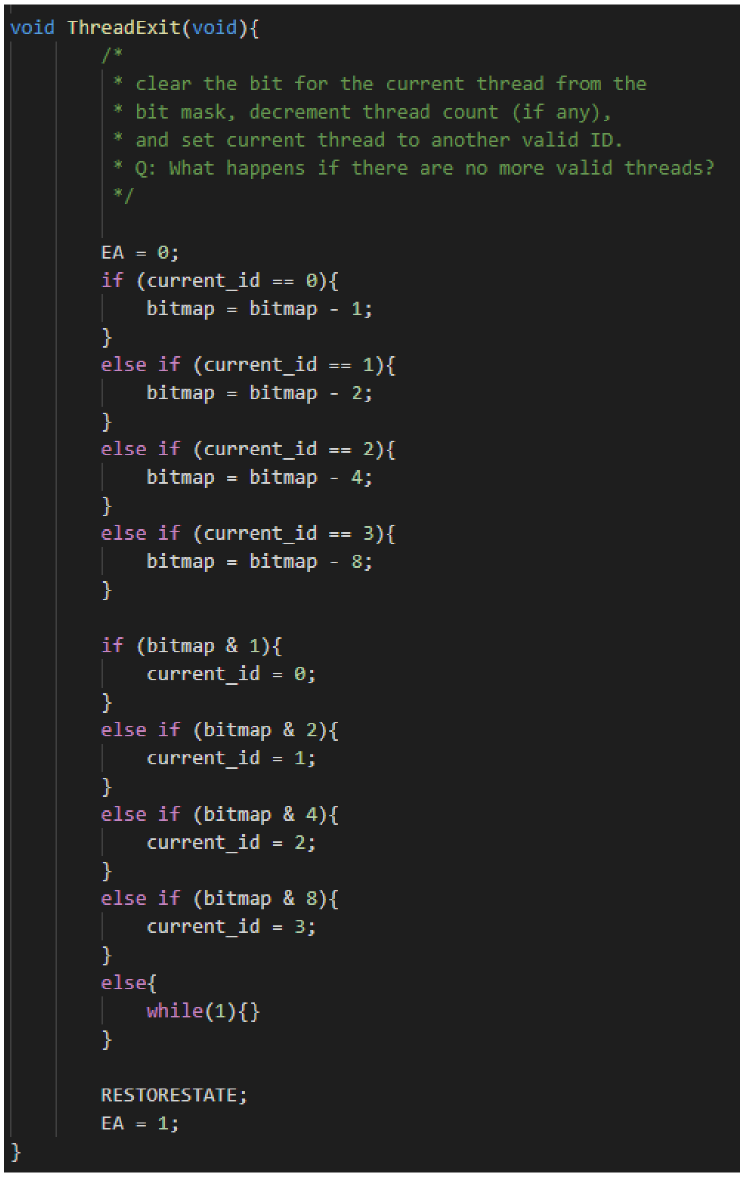
IPNS 106010018 Yen-Fong Li

It is made up by three parts of files:

1. preemptive.c
2. preemptive.h
3. testparkig.c

preemptive.c

ThreadExit()



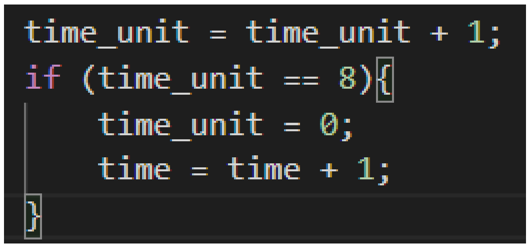
```
void ThreadExit(void){
        /*
         * clear the bit for the current thread from the
         * bit mask, decrement thread count (if any),
         * and set current thread to another valid ID.
         * Q: What happens if there are no more valid threads?
         */

        EA = 0;
        if (current_id == 0){
            bitmap = bitmap - 1;
        }
        else if (current_id == 1){
            bitmap = bitmap - 2;
        }
        else if (current_id == 2){
            bitmap = bitmap - 4;
        }
        else if (current_id == 3){
            bitmap = bitmap - 8;
        }

        if (bitmap & 1){
            current_id = 0;
        }
        else if (bitmap & 2){
            current_id = 1;
        }
        else if (bitmap & 4){
            current_id = 2;
        }
        else if (bitmap & 8){
            current_id = 3;
        }
        else{
            while(1){}
        }

        RESTORESTATE;
        EA = 1;
}
```

We first decided which thread_id are we going to exit. Then minus the bitmap based on it.

Then We have to decide which thread_id should be the next.

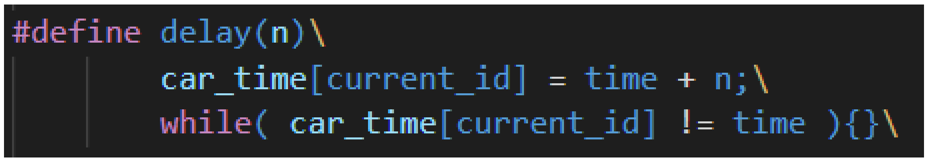
mytimer0handler()

All things are like things before except we should count the time in mytimer0handler() so that we can implement the delay() function.



```
time_unit = time_unit + 1;
if (time_unit == 8){
    time_unit = 0;
    time = time + 1;
}
```

preemptive.h



```
#define delay(n)\
        car_time[current_id] = time + n;\
        while( car_time[current_id] != time ){}\
```

In delay function, we use an array (car_time) to record. Everytime a car is in a parking lot and call the delay function, we will add the time now by n. And then use a while loop to check if the time is achieve the delay we set.

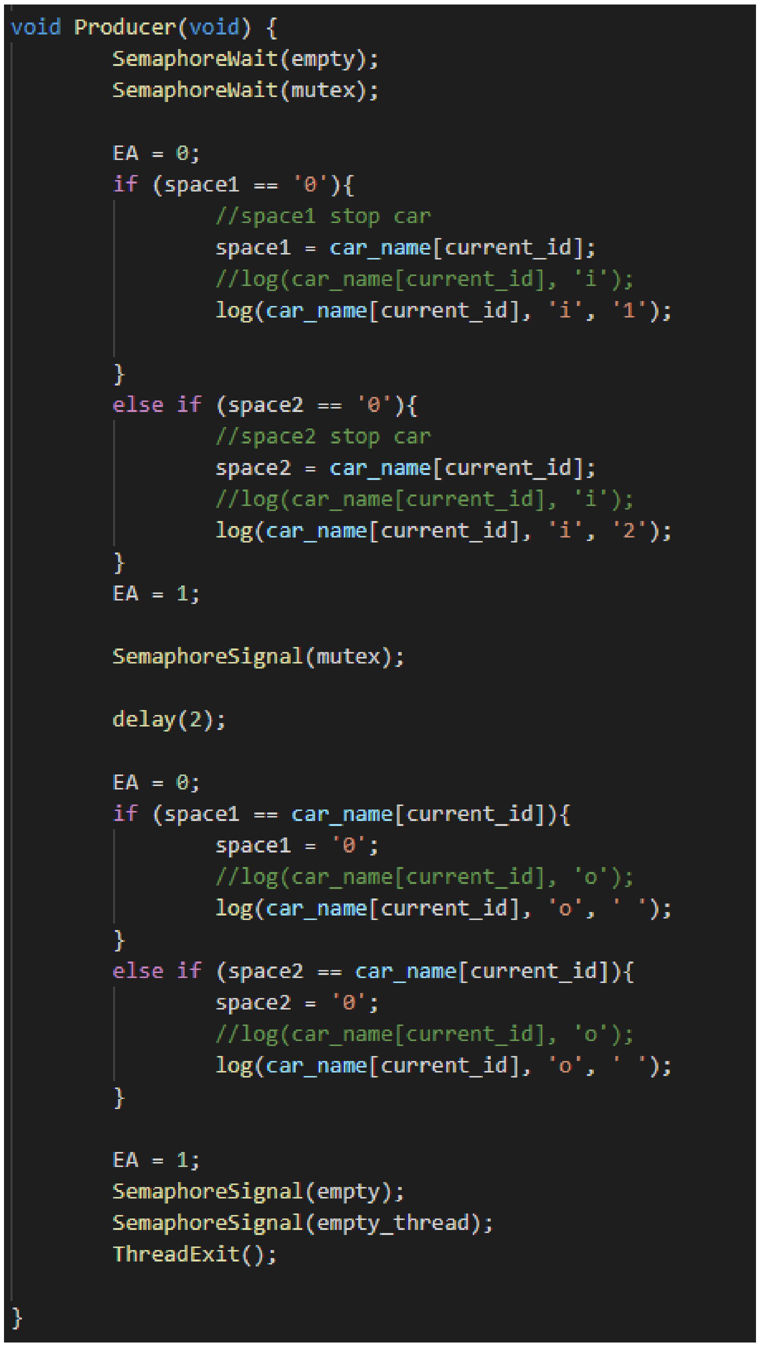
testparking.c

This is the part that change the most.

First, we don't need Consumer at all.

Next, the Producer() function should be modified.

Producer()



```
void Producer(void) {
        SemaphoreWait(empty);
        SemaphoreWait(mutex);

        EA = 0;
        if (space1 == '0'){
                //space1 stop car
                space1 = car_name[current_id];
                //log(car_name[current_id], 'i');
                log(car_name[current_id], 'i', '1');

        }
        else if (space2 == '0'){
                //space2 stop car
                space2 = car_name[current_id];
                //log(car_name[current_id], 'i');
                log(car_name[current_id], 'i', '2');
        }
        EA = 1;

        SemaphoreSignal(mutex);

        delay(2);

        EA = 0;
        if (space1 == car_name[current_id]){
                space1 = '0';
                //log(car_name[current_id], 'o');
                log(car_name[current_id], 'o', ' ');
        }
        else if (space2 == car_name[current_id]){
                space2 = '0';
                //log(car_name[current_id], 'o');
                log(car_name[current_id], 'o', ' ');
        }

        EA = 1;
        SemaphoreSignal(empty);
        SemaphoreSignal(empty_thread);
        ThreadExit();

}
```

The semaphore empty is used to checking whether there is still a parking lot there.

We use two variables – space1 and space2 to imply whether the parking lot is empty or not (0 for empty)

If the parking lot is empty, we store a car in it and change space1 to the car's name, which we recorded by and array car_name.

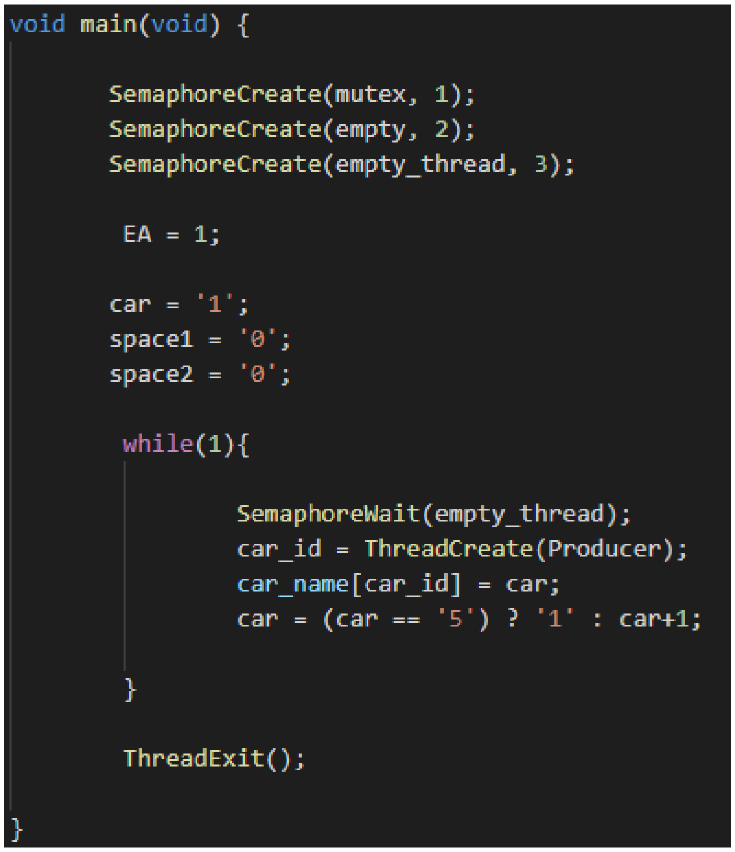
Then delay for 2.

Below we have to check which parking lot is the car in. Then we move out the car (set space1 to 0)

Last, we signal empty.

The semaphore empty_thread is used to record how many threads are there. We have MAXTHREADS 4, one is used for main. So we have three threads for the cars.

main()


```
void main(void) {

      SemaphoreCreate(mutex, 1);
      SemaphoreCreate(empty, 2);
      SemaphoreCreate(empty_thread, 3);

       EA = 1;

      car = '1';
      space1 = '0';
      space2 = '0';

       while(1){

              SemaphoreWait(empty_thread);
              car_id = ThreadCreate(Producer);
              car_name[car_id] = car;
              car = (car == '5') ? '1' : car+1;

       }

       ThreadExit();

}
```

We create three semaphores.

Then initialize some variables.

Then in a while loop, We start to produce a car and record it's car's name.

Questions

- **what does your timer-0 ISR have to do to support these multiple delays and now()?**

  We have to use it as time_unit and create our own time.

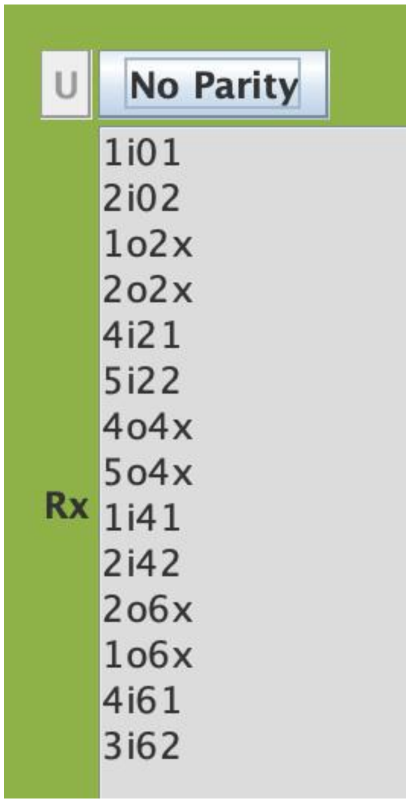
- **what if all threads call delay() and happen to finish their delays all at the same time? How can you ensure the accuracy of your delay? (i.e., between *n* and *n*+0.5 time units)?**

  It will print first, however it is still at the same time.

- **How does the worst-case delay completion (i.e., all threads finish delaying at the same time) affect your choice of time unit?**

  It cannot be too small.

Results



```
1i01
2i02
1o2x
2o2x
4i21
5i22
4o4x
5o4x
1i41
2i42
2o6x
1o6x
4i61
3i62
```