OS checkpoint 5

107021121 馮云菲

1. delay(n) function and now() function:

• delay() function

Requirement: delay() is not an exact delay but is a delay for “at least *n* time units” and “less than (*n* + 0.5) time units”

Suppose time unit is equal to x*time quantum, I.e. one time unit can execute x threads.

Each ISR will need time quantum to handle a thread.

In order to wake up someone in (n+0.5) time units, we will need to set time quantum at least equal to 2x times of time unit. (1/0.5 = 2)

Here, we have 4 threads.

=> Time unit at least 8(2*4) times of time quantum,.

Time quantum in my code is called timer.

```
17  #define delay(n)\
18          time_temp[cur_thread] = time + n;\
19          while( time_temp[cur_thread] != time ){}\
20
```

In myTimer0Handler() function:

```
timer = timer + 1;  // enter ISR then add 1
if(timer == 8){
    time = time + 1;
    timer = 0;
}
```

Time increase 1 if execute 8 times of ISR.

• now () function:

```
65  // return current time
66  unsigned char now(void){
67      return time;
68  }
69
```

2.Robust Thread Termination and Creation

- ThreadExit():
  When entering ThreadExit(), it would delete the thread by updating bitmap(i.e. to mark the thread as unallocated) and find the next thread to run.
  Also, increasing time when entering this function.

The thread needs to be deleted is the current thread.

```
void ThreadExit(void) {
        EA = 0;
        // delete thread by clearing the bit for the current thread from the bit mask ?
        // if cur_thread = 0, mask -=0b0001; cur_thread = 1, mask-=0b0010 ... ?

        if(cur_thread == 0) mask = mask - 1;
        else if( cur_thread == 1 )mask = mask - 2;
        else if( cur_thread == 2 )mask = mask - 4;
        else if( cur_thread == 3 )mask = mask - 8;

        if(  mask & 1  ){
            cur_thread = 0;
        }else if(  mask & 2  ){
            cur_thread = 1;
        }else if(  mask & 4  ){
            cur_thread = 2;
        }else if(  mask & 8  ){
            cur_thread = 3;
        }else{
            //RESTORESTATE;
            while(1){}
        }

        RESTORESTATE;
        EA = 1;
}
```

If the thread finishes, then the thread should delete in order to let other thread to be created.

- We need to utilize semaphore in main() function to limit threads to 4. We create three cars in three different threads.

```
// create three cars in three threads first
car_id = ThreadCreate( Producer );
car_name[ car_id ] = car;
car = car+1;

car_id = ThreadCreate( Producer );
car_name[ car_id  ] = car;
car = car+1;

car_id = ThreadCreate( Producer );
car_name[ car_id  ] = car;
car = car+1;
```

- We use semaphore to wait until some cars leaves. Moreover, we need to enable interrupt before it, and disable after it.

- In Producer()
  When the car leaves the parking lot, signal the semaphore next_car and run ThreadExit() to delete this thread.

```
while(1){
    EA = 1; // enable interrupt
    SemaphoreWaitBody(next_car, L(__COUNTER__) );
    EA = 0;

    car_id = ThreadCreate( Producer );
    // if( car == Token2 || car == Token ){
    //     car = (car == '5') ? '1' : car+1;
    // }
    car_name[ car_id  ] = car;

    car = (car == '5') ? '1' : car+1;
}
ThreadExit();
```

```
SemaphoreSignal(empty);
SemaphoreSignal(next_car);
ThreadExit();       //delete thread
```