105021108 謝佳勳

1. delay(n), now()

   (i) now()

```
unsigned char now(void) { return time; }
```

   I use a char time to record the current time.

```
void myTimer0Handler(void){
//……//
counter = (counter==4) ? 0 : counter+1;
if(!counter) time++;
//……//
}
```

   And update the time when counter count 5 times, where counter count when the timer0 interrupt happen.

   In my project, I use a thread for thread manager, so there are at most 3 threads calling delay(n), if all 3 threads end delay at the same time, the worst case is “thread1 occupy the CPU all the 1st timer0 cycle, then context switch such that thread2 occupy the CPU all the 2nd timer0 cycle, finally context switch to thread3”. So all the threads will get correct delay in the sense that delay for “at least n time unit” and “less than (n+0.5) time unit”.

   (ii) delay(n)

```
void delay(unsigned char n) {
D[ID] = now() + n;
bitmap[ID] = -2;
ThreadYield();
}
```

   When doing delay, I use a char D[4] to record what the time this thread will be recovered and then I set the bitmap[ID] to -2 to indicate that the thread is being delay. After that I call the threadyield() to context switch to other thread.

2. robust thread termination and creation

   I do push the address of threadexit() on the stack, so when the thread return, it will enter in the threadexit(), then it will terminate fine.

```
ThreadID ThreadCreate(FunctionPtr fp) {
//……//
__asm
mov a,DPL
mov b,DPH
mov dptr,#_ThreadExit
```

```
push DPL
push DPH
push a
push b
__endasm;
//……//
}
```

And I use a semaphore to control the total number of threads.

```
void Bootstrap(void) {
SemaphoreCreate(thread, 4);
__asm
mov _th_tail,#0x7C
__endasm;
}
ThreadID ThreadCreate(FunctionPtr fp) {
SemaphoreWait(thread,th_tail);
//……//
}
void ThreadExit(void) {
SemaphoreSignal(thread,th_tail);
//……//
}
```

Because the thread manager is also a thread , when all other threads exits, it will always in the thread manager, so enter in an infinite loop.

3. parking lot example

(i) log

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	4B	27	00	00	00	00	00	00	25	6B	00	00	00	00	00	30
10	26	5D	00	00	00	00	00	03	27	4C	00	00	00	00	FF	06
20	48	58	68	78	01	00	00	00	00	0A	00	00	04	00	06	05
30	01	02	81	83	02	03	03	06	83	85	01	03	6C	7C	00	10
40	09	0B	F6	08	01	80	00	00	01	00	85	01	07	0A	00	00
50	09	0B	41	0A	25	04	BC	05	09	03	09	00	05	03	00	00
60	09	0B	08	04	26	87	03	00	11	00	00	18	5C	00	00	00
70	09	0B	0E	08	31	01	00	00	19	00	00	00	00	01	00	00

My log is at 0x30~0x39 :

01	02	81	83	02	03	03	06	83	85

This means:

Car 1 in	Car 1 out	Car 2 in	Car 2 out	Car 3 in	Car 3 out	Car 4 in	Car 4 out	Car 5 in	Car 5 out

Each data $(\text{abcdefgh})_2$ means “plot (a+1) at time $(\text{bcdefgh})_2$”

So, the above log means:

Car 1 in plot1 at time 1.

Car 1 out plot1 at time 2.

Car 2 in plot2 at time 1.

Car 2 out plot2 at time 3.

Car 3 in plot1 at time 2.

Car 3 out plot1 at time 3.

Car 4 in plot1 at time 3.

Car 4 out plot1 at time 6.

Car 5 in plot2 at time 3.

Car 5 out plot2 at time 5.

(ii) Extra credit: display the output of my log to UART in a human-readable text format.

Because my log is not arranged by time, we must print the log after all cars are leave. So I use a semaphore for print:

For all 5 cars:

```
void Cari(void) {
    SemaphoreWait(mutex,m_tail);
     //log car i in which plot at what time//
    SemaphoreSignal(mutex,m_tail);
    SemaphoreSignal(print,p_tail);
    //delay//
    SemaphoreWait(mutex,m_tail);
     //log car i out which plot at what time//
    SemaphoreSignal(mutex,m_tail);
    SemaphoreSignal(print,p_tail);
}
```

And for print log:

```
void Printer(void) {
     //initialize UART//
    SemaphoreWait(print,p_tail);
    SemaphoreWait(print,p_tail);
    SemaphoreWait(print,p_tail);
    SemaphoreWait(print,p_tail);
    SemaphoreWait(print,p_tail);
    SemaphoreWait(print,p_tail);
    SemaphoreWait(print,p_tail);
```

```
    SemaphoreWait(print,p_tail);
    SemaphoreWait(print,p_tail);
    SemaphoreWait(print,p_tail);
     //do print log//
}
```

Result: