```
PS C:\Users\josh9\OneDrive\桌面\大三上\OS\109062119-ppc5> make clean
del *.hex *.ihx *.lnk *.lst *.map *.mem *.rel *.rst *.sym *.asm *.lk
PS C:\Users\josh9\OneDrive\桌面\大三上\OS\109062119-ppc5> make
sdcc -c  testparking.c
testparking.c:46: warning 158: overflow in implicit constant conversion
testparking.c:49: warning 158: overflow in implicit constant conversion
testparking.c:58: warning 158: overflow in implicit constant conversion
testparking.c:61: warning 158: overflow in implicit constant conversion
testparking.c:79: warning 158: overflow in implicit constant conversion
testparking.c:82: warning 158: overflow in implicit constant conversion
testparking.c:91: warning 158: overflow in implicit constant conversion
testparking.c:94: warning 158: overflow in implicit constant conversion
testparking.c:112: warning 158: overflow in implicit constant conversion
testparking.c:115: warning 158: overflow in implicit constant conversion
testparking.c:124: warning 158: overflow in implicit constant conversion
testparking.c:127: warning 158: overflow in implicit constant conversion
testparking.c:145: warning 158: overflow in implicit constant conversion
testparking.c:148: warning 158: overflow in implicit constant conversion
testparking.c:157: warning 158: overflow in implicit constant conversion
testparking.c:160: warning 158: overflow in implicit constant conversion
testparking.c:178: warning 158: overflow in implicit constant conversion
testparking.c:181: warning 158: overflow in implicit constant conversion
testparking.c:190: warning 158: overflow in implicit constant conversion
testparking.c:193: warning 158: overflow in implicit constant conversion
sdcc -c  preemptive.c
preemptive.c:162: warning 85: in function ThreadCreate unreferenced function argument : 'fp'
sdcc  -o testparking.hex testparking.rel preemptive.rel
```

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

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

- Based on the above requirement, state your choice of time unit and provide your justification for how you think you can implement a `delay()` that meets the requirement above.
- what does your timer-0 ISR have to do to support these multiple delays and now()?
- Count the time unit.
- 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)?
- EA = 0; disable all interrupt.
- How does the worst-case delay completion (i.e., all threads finish delaying at the same time) affect your choice of time unit?
- Too small unit will have wrong answer.

```
void ThreadExit(void) {
    __critical{
     temp = 1 << cur_thread;
     mask ^= temp;
     for(i=0 ; i < 4 ; i++) {
       temp = 1 << i;
       if( mask & temp ){
          cur_thread = i;
          break;
       }
     }
       if(i == 4) {
          while(1){}
       }

       RESTORESTATE;
    }
}
```

Car, in/out, spot1/2, time
1i10
2i20
2o22
1o12
4i12

3i22
4o14
3o24
5i14
1i24
5o16
1o26
2i16
2o18