

Part One: $\text{delay}(n), \text{now}()$

- what does your timer-0 ISR have to do to support these multiple delays and $\text{now}()$?

When each time $\text{myTimer0Handler}(\text{void})$ is called, cnt will be incremented by 1. Every four times $\text{myTimer0Handler}(\text{void})$ is called, now will be incremented by 1 and cnt will be flushed to 0.

- what if all threads call $\text{delay}()$ 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)?

-

Instead of using the real time unit, we set a logically time unit. Because the time we use at floating point might be regarded as the same number. But if we use logically time units, we set we don't have to worry about that kind of issue.

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

The time unit I choose is related to the interrupt interval of switching threads but not actual time units. The length of one time unit in my project equals to four interrupt intervals of timer0. I can set every cnt to be $1/8$ time units and make now to increment every 8 interrupts. Then, we do nothing when cnt is between 5 and 8 to make $\text{delay}(n)$ bounds to "less than $(n+0.5)$ time units, yet that would be less efficient.

Part Two: Robust Thread Termination and Creation

```
void ThreadExit(void) {
    EA=0; // disable interrupt when terminate thread
    __asm // reset stack to 0s (for debug)
    mov A, #0 // >> so we can clearly see whether
    push A // >> a thread is terminated or not
    push A
    push A
    push A
    push A
    push A
    push A
    push A
    __endasm;

    sp[curThread] = 0; // reset stack pointer
    bitmap &= ~(1<<curThread); // update bitmap
    if(bitmap == 0){ // if it is the last thread
        while(1){}; // then enter infinite loop
    }

    do { // switch to the next ready thread
        curThread = (curThread == MAXTHREADS-1) ? 0 : curThread+1;
        if(bitmap & checkAlive[curThread]) break;
    } while (1);
    RESTORESTATE;
    EA=1; // enable interrupt after terminate and switch thread
}
```

Part Three: Parking Lot Example

Car1 park at spot 0 at time unit: 0 and exits at time unit: 4
Car2 park at spot 1 at time unit: 2 and exits at time unit: 6
Car3 park at spot 0 at time unit: 4 and exits at time unit: 8
Car4 park at spot 1 at time unit: 6 and exits at time unit: 10
Car5 park at spot 0 at time unit: 8 and exits at time unit: 12

Part Four: Typescript and screenshots

```
project5 — -zsh — 86x26

[zhangjunteng@zhangjuengdeAir project5 % ls
Makefile      preemptive.c    preemptive.h    testparking.c
[zhangjunteng@zhangjuengdeAir project5 % make
sdcc -c testparking.c
testparking.c:196: warning 158: overflow in implicit constant conversion
sdcc -c preemptive.c
preemptive.c:86: warning 85: in function ThreadCreate unreferenced function argument :
'fp'
sdcc -o testparking.hex testparking.rel preemptive.rel
[zhangjunteng@zhangjuengdeAir project5 % ls
Makefile      preemptive.lst    testparking.asm testparking.lst testparking.rst
preemptive.asm preemptive.rel    testparking.c   testparking.map testparking.sym
preemptive.c   preemptive.rst    testparking.hex testparking.mem
preemptive.h   preemptive.sym    testparking.lk  testparking.rel
zhangjunteng@zhangjuengdeAir project5 %
```

data memory

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
2		sem	spotsSync	spots[2]		CarIn[5]							delayid[MAXTHREADS]				
3	sp[MAXTHREADS]			curThread	bitmap	newThread	i	tmp	tmp2	checkAlive[MAXTHREADS]			cnt		now		
4	stack space for thread 0										CarOut[5]						
5	stack space for thread 1										CarSpot[5]						
6	stack space for thread 2																
7	stack space for thread 3																