

OS project checkpoint 5

105021120 朱世耘

function address:

	Value	Global	Global Defined In Module
C:	00000014	_Car1	testparking
C:	00000051	_Car2	testparking
C:	0000008E	_Car3	testparking
C:	000000CB	_Car4	testparking
C:	00000108	_Car5	testparking
C:	00000145	_output	testparking
C:	00000BEB	_main	testparking
C:	00000C18	__sdcc_gsinit_startup	testparking
C:	00000C1C	__mcs51_genRAMCLEAR	testparking
C:	00000C1D	__mcs51_genXINIT	testparking
C:	00000C1E	__mcs51_genXRAMCLEAR	testparking
C:	00000C1F	_timer0_ISR	testparking
C:	00000C23	_now	preemptive
C:	00000C27	_delay	preemptive
C:	00000C47	_thread_manager	preemptive
C:	00000C7A	_myTimer0Handler	preemptive
C:	00000CD2	_Bootstrap	preemptive
C:	00000D1E	_ThreadCreate	preemptive
C:	00000D84	_ThreadYield	preemptive
C:	00000DCF	_ThreadExit	preemptive

data address:

	Value	Global	Global Defined In Module
	00000000	._.ABS.	preemptive
	00000020	_mutex	testparking
	00000021	_thread_count	preemptive
	00000022	_parklot	testparking
	00000024	_bitmap	preemptive
	00000029	_manager_ID	preemptive
	0000002A	_tmp1	preemptive
	0000002B	_tmp2	preemptive
	0000002C	_i	preemptive
	0000002D	_count	preemptive
	0000002E	_x	testparking
	00000030	_time	preemptive
	00000031	_log	testparking
	0000004A	_ID	preemptive
	0000005C	_ssp	preemptive
	0000006C	_d_time	preemptive
	0000007D	_printer	testparking
	0000007E	_cnt	preemptive
	0000007F	_var	preemptive

log[15] : from 31 to 3F, record every cars' arrived/leave/location

time : 30, increase when thread(main) get the power

parklot[2] : 22,23, record which car is in the parking lot now

bitmap[4] : from 24 to 27, record every thread status 0:empty 1:using 7:delaying

key function:

now() and delay()

```
unsigned char now(void){
    return time;
    // time++ when thread[1](main) get the power
}
void delay(unsigned char n){
    d_time[ID]=now()+n;
    // set bitmap[]=7 represent delay
    bitmap[ID]=7;
    ThreadYield();
}
```

TimerHandler()

```
void myTimerHandler(void){
    SAVESTATE;
    // check if any delay thread times up
    for (i=0; i<MAXTHREADS; i++) {
        if ((bitmap[i]==7) && (d_time[i]==time)) {
            bitmap[i]=1;
        }
    }
    tmp1 = ID;
    ID = manager_ID;
    RESTORESTATE;
    ID = tmp1;
}
```

ThreadYield()

```
void ThreadYield(void) {
    EA = 0;
    SAVESTATE;
    // find next available thread
    do {
        ID++;
        ID = ID % MAXTHREADS;
        if(bitmap[ID]==1){
            break;
        }
    } while (1);
    // check if time should increase
    if (ID==1) {
        cnt++;
        if (cnt==3) {
            cnt=0;
            time++;
        }
    }
    RESTORESTATE;
    EA = 1;
}
```

time++ when thread(main) get power every three times

Car1(), same as other cars

```
// Car 1 to 5, find empty parklot, record in log[], leave
void Car1(void) {
    log[0] = now();
    SemaphoreWait(mutex);
    if (parklot[0]==0) {
        parklot[0]=1;
        log[2]=0;
    }
    else{
        parklot[1]=1;
        log[2]=1;
    }
    SemaphoreSignal(mutex);
    delay(2);
    log[1] = now();
    parklot[(log[2])]=0;
    SemaphoreSignal(printer);
    ThreadExit();
}
```

output()

```
// output log[]
void output(){
    // wait until five cars have leaved
    SemaphoreWait(printer);
    SemaphoreWait(printer);
    SemaphoreWait(printer);
    SemaphoreWait(printer);
    SemaphoreWait(printer);

    TMOD |= 0x20;
    TH1 = -6;
    SCON = 0x50;
    TR1 = 1;

    // output car i:arrive/leave/location
    print('\0');
    print('a');
```

main()

```
void main(void) {
    // initialize Semaphore and parklot
    SemaphoreCreate(mutex, 1);
    SemaphoreCreate(printer, 0);
    parklot[0]=0;
    parklot[1]=0;

    ThreadCreate(Car1);
    ThreadCreate(Car2);
    ThreadCreate(Car3);
    ThreadCreate(Car4);
    ThreadCreate(Car5);
    output();
}
```

execution:

Let car1 delay(2), car2 delay(3), car3 delay(5), car4 delay(3), car5 delay(6)

therefore, two parking lots should be:

parklot[0]: 1~3 car1, 3~8 car3

parklot[1]: 1~4 car2, 4~7 car4, 7~13 car5

System Clock (MHz) 11.0592 2000 Update Freq.

8051

PC 0x0C72

Modify RAM

Data Memory

addr	0x00	0x01	0x02	0x03	0x04	0x05	0x06	0x07	0x08	0x09	0x0A	0x0B	0x0C	0x0D	0x0E	0x0F
00	5C	5C	00	00	00	00	00	01	25	5C	00	00	00	00	31	01
10	26	5C	00	00	00	00	00	05	27	5C	00	00	00	00	00	06
20	01	03	00	00	01	00	00	00	00	00	00	18	04	00	01	00
30	13	01	03	00	01	04	01	03	08	00	04	07	01	07	03	01
40	49	0C	78	0C	00	00	00	00	00	00	00	00	00	00	00	00
50	4C	0B	33	80	03	01	08	01	08	00	00	00	48	46	66	76
60	BB	00	26	00	03	00	11	00	00	00	00	00	00	00	08	0D
70	35	01	27	00	07	00	18	00	00	00	00	00	00	00	00	01

Copyright © 2005–2016 James Rogers Remove All Breakpoints

Assembly Code:

```
0C3F | ADD A,#24H
0C41 | MOV R0,A
0C42 | MOV @R0,#07H
0C44 | LJMP 0D84H
0C47 | INC 4AH
0C49 | ANL 4AH,#03H
0C4C | MOV A,4AH
0C4E | ADD A,#24H
0C50 | MOV R1,A
0C51 | MOV 07H,@R1
0C53 | CJNE R7,#01H,0F1H
0C56 | MOV A,#01H
0C58 | CJNE A,4AH,0CH
0C5B | INC 7EH
0C5D | MOV A,#03H
0C5F | CJNE A,7EH,05H
0C62 | MOV 7EH,#00H
0C65 | INC 30H
0C67 | MOV A,4AH
0C69 | ADD A,#5CH
0C6B | MOV R1,A
0C6C | MOV 81H,@R1
0C6E | POP 0D0H
0C70 | POP 83H
0C72 | POP 82H
```

UART Configuration:

U No Parity 8-bit UART @ 4800 Baud

Rx arrive/leave/space

Rx Reset

Tx car1:1/3/0

Tx car2:1/4/1

Tx car3:3/8/0

Tx car4:4/7/1

Tx car5:7/13/1

Tx Send

ox31 to ox3F store log[15]

arrive/leave/space

car1: 1 3 0

car2: 1 4 1

car3: 3 8 0

car4: 4 7 1

car5: 7 13 1

Observe semaphore:

there are three semaphore,

mutex: at 20, initial to be 1, change when enter or leave critical section

thread_count: at 21, initial to be 4, change when threadcreate and threadexit

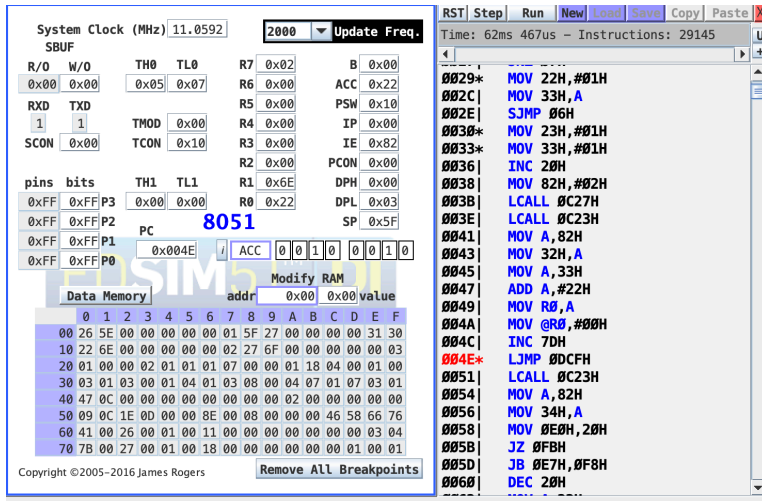
printer: at 7D, initial to be 0, increase when car leave, decrease when outputting

```
// Car 1 to 5, find empty parklot, record in log[], leave
void Car1(void) {
    log[0] = now();
    SemaphoreWait(mutex);
    if (parklot[0]==0) {
        parklot[0]=1;
        log[2]=0;
    }
    else{
        parklot[1]=1;
        log[2]=1;
    }
    SemaphoreSignal(mutex);
    delay(2);
    log[1] = now();
    parklot[(log[2])]=0;
    SemaphoreSignal(printer);
    ThreadExit();
}
```

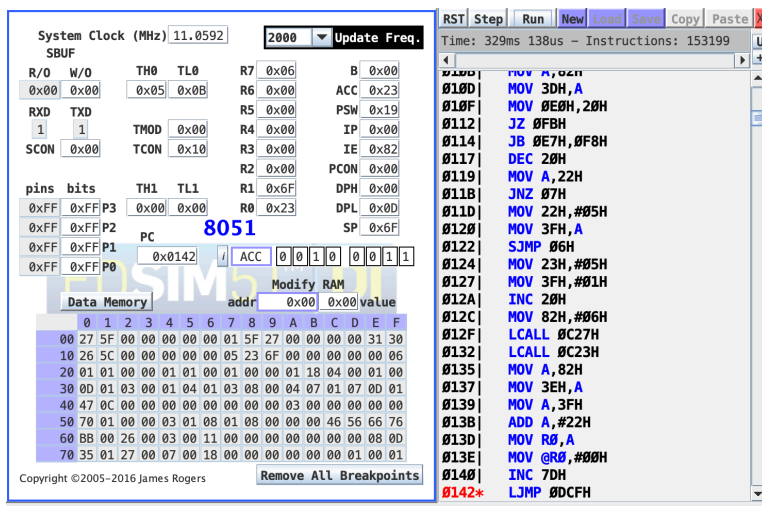
Car1 is doing critical section, mutex=0

The screenshot displays the Proteus ISIS simulation environment. On the left, the 8051 microcontroller's internal registers and memory are visible. The PC (Program Counter) is highlighted at address 8051, containing the value 0x0029. The ACC (Accumulator) is at 0x0000. The RAM memory is shown at the bottom, with addresses 0 to 70. On the right, the assembly code is displayed, showing the current instruction at address 0029: `MOV 22H, #01H`. The code includes various instructions such as `LJMP`, `RETI`, `LCALL`, `MOV`, `JZ`, `JB`, `DEC`, `INC`, and `LCALL`. The status bar at the bottom indicates the copyright information: Copyright © 2005–2016 James Rogers.

Car1 finished, printer++



all the car have leaved, printer is large enough to output



outputing, printer=o

