

Report Checkpoint 2

Jonathan Sutedjo 鄭安良 111006207

```
/cygdrive/c/Users/Jonat/Documents/!NTHU Classes/Operating System/Fin Pr/J...
preemptive.c:274: warning 85: in function ThreadCreate unreferenced function arg
ument : 'fp'
sdcc -o testpreempt.hex testpreempt.rel preemptive.rel

Jonat@LAPTOP-7F30BD0F /cygdrive/c/Users/Jonat/Documents/!NTHU Classes/Operating
System/Fin Pr/Jon/ppc2
$ make clean
rm *.hex *.ihx *.lnk *.lst *.map *.mem *.rel *.rst *.sym *.asm *.lk
rm: cannot remove '*.ihx': No such file or directory
rm: cannot remove '*.lnk': No such file or directory
make: *** [Makefile:25: clean] Error 1

Jonat@LAPTOP-7F30BD0F /cygdrive/c/Users/Jonat/Documents/!NTHU Classes/Operating
System/Fin Pr/Jon/ppc2
$ make
sdcc -c testpreempt.c
sdcc -c preemptive.c
preemptive.c:274: warning 85: in function ThreadCreate unreferenced function arg
ument : 'fp'
sdcc -o testpreempt.hex testpreempt.rel preemptive.rel

Jonat@LAPTOP-7F30BD0F /cygdrive/c/Users/Jonat/Documents/!NTHU Classes/Operating
System/Fin Pr/Jon/ppc2
$
```

```

void Producer(void)
{
    /*
     * [TODO]
     * initialize producer data structure, and then enter
     * an infinite loop (does not return)
     */
    __critical{
        SharedBuffer = 'A'-1;
    }
    while (1)
    {
        /* [TODO]
         * wait for the buffer to be available,
         * and then write the new data into the buffer */
        if(Buffer_Availability){
        }
        else{
            __critical{
                Buffer_Availability += 1;
            }
            if(SharedBuffer == 'Z'){
                __critical{
                    SharedBuffer = 'A';
                }
            }
            else{
                __critical{
                    SharedBuffer += 1;
                }
            }
        }
    }
}

void Consumer(void)
{
    /*
     * [TODO]
     * initialize Tx for polling
     */
    TMOD |= 0x20;
    TH1 = (char)-6;
    SCON = 0x50;
    TR1 = 1;
    TI = 1;

    while (1)
    {
        /*
         * [TODO]
         * wait for new data from producer
         * write data to serial port Tx,
         * poll for Tx to finish writing (TI),
         * then clear the flag
         */
        if(Buffer_Availability){
            while (!TI){
            }
            SBUF = SharedBuffer;
            TI = 0;
            __critical{
                Buffer_Availability -= 1;
            }
        }
    }
}

```

Above are my producer and consumer, they have the same functionality as checkpoint 1 but the changes that I made are the addition of removing the “ThreadYield()” and adding the wrapper “__critical” to some parts of my code. I also made a change to the consumer function where I change the “TMOD = 0x20” to become “TMOD |= 0x20”.

In the producer part, it is responsible for generating the characters in a sequential order from ‘A’ to ‘Z’ and storing it in the shared buffer, meanwhile in the consumer part, it continues to retrieve character from the shared buffer and writes them to the serial port.

```

#define SAVESTATE \
{ \
    __asm \
    PUSH ACC \
    PUSH B \
    PUSH DPL \
    PUSH DPH \
    PUSH PSW \
    __endasm; \
    switch (curThreadID) { \
        case '0': \
            __asm \
            MOV 0x30, SP \
            __endasm; \
            break; \
        case '1': \
            __asm \
            MOV 0x31, SP \
            __endasm; \
            break; \
        case '2': \
            __asm \
            MOV 0x32, SP \
            __endasm; \
            break; \
        case '3': \
            __asm \
            MOV 0x33, SP \
            __endasm; \
            break; \
        default: \
            break; \
    } \
} \

#define RESTORESTATE \
{ \
    switch (curThreadID) { \
        case '0': \
            SP = Pointer[0]; \
            break; \
        case '1': \
            SP = Pointer[1]; \
            break; \
        case '2': \
            SP = Pointer[2]; \
            break; \
        case '3': \
            SP = Pointer[3]; \
            break; \
        default: \
            break; \
    } \
    __asm \
    POP PSW \
    POP DPH \
    POP DPL \
    POP B \
    POP ACC \
    __endasm; \
} \

```

Above are my savestate and restorestate function where savestate saves the state of the current thread by pushing the key CPU registers (ACC, B, DPL, DPH, and PSW) onto the stack and storing the stack pointer into the array indexed by the current thread ID, and where restorestate restores the state of the current thread by loading the saved stack pointer from the array and popping the key CPU registers (PSW, DPL, DPH, B, and ACC) from the stack.

```

void Bootstrap(void)
{
    /*
     * [TODO]
     * initialize data structures for threads (e.g., mask)
     *
     * optional: move the stack pointer to some known location
     * only during bootstrapping. by default, SP is 0x07.
     *
     * [TODO]
     *     create a thread for main; be sure current thread is
     *     set to this thread ID, and restore its context,
     *     so that it starts running main().
     */

    ValidBitMap = 0b0000;
    Pointer[0] = 0x3F;
    Pointer[1] = 0x4F;
    Pointer[2] = 0x5F;
    Pointer[3] = 0x6F;
    TMOD = 0; //timer zero
    IE = 0x82;
    TR0 = 1;
    curThreadID = ThreadCreate(main);
    RESTORESTATE;
}

```

For my bootstrap, I initialized the TMOD, IE, and TR0 here and make the TMOD = 0 while the TR0 and IE stays the same. The other part of the code here, I do not make any change and still the same with checkpoint 1. The change is only the “TMOD = 0”.

```

void myTimer0Handler(void){
    EA = 0;
    SAVESTATE;
    do{
        curThreadID = (curThreadID == '3') ? '0' : curThreadID + 1;

        switch (curThreadID) {
            case '0':
                if ((ValidBitMap & 0b0001) == 0b0001) break;
                continue;
            case '1':
                if ((ValidBitMap & 0b0010) == 0b0010) break;
                continue;
            case '2':
                if ((ValidBitMap & 0b0100) == 0b0100) break;
                continue;
            case '3':
                if ((ValidBitMap & 0b1000) == 0b1000) break;
                continue;
        }
        break;
    } while (1);
    RESTORESTATE;

    EA = 1;
    __asm
    RETI
    __endasm;
}

```

This is my myTimer0Handler where I implement something like ThreadYield() that I removed previously in some parts of my code for this part 2. The difference between this function with the ThreadYield is where I use RETI instead of RET.

Other part of my code are still the same with previous, the other changes I made is in the makefile because of changing the file name. Other than that the header also as I made changes to the file name.

	Value	Global	Global Defined In Module
C:	00000014	_Producer	testpreempt
C:	00000065	_Consumer	testpreempt
C:	00000096	_main	testpreempt
C:	000000A7	__sdcc_gsinit_startup	testpreempt
C:	000000AB	__mcs51_genRAMCLEAR	testpreempt
C:	000000AC	__mcs51_genXINIT	testpreempt
C:	000000AD	__mcs51_genXRAMCLEAR	testpreempt
C:	000000AE	_timer0_ISR	testpreempt
C:	000000B2	_Bootstrap	preemptive
C:	00000106	_ThreadCreate	preemptive
C:	000001BA	_ThreadYield	preemptive
C:	0000029A	_ThreadExit	preemptive
C:	00000381	_myTimer0Handler	preemptive

This is some content of my testpreempt.map file where it shows addresses.

EdSim51DI - Version 2.1.38 & Dynamic Interface x | testpreempt.hex

System Clock (MHz): 11.0592 | Update Freq. 1

RST Step Run New Load Save CPY Paste BP

Time: 25us - Instructions: 14

00E2 | SJMP 0FH

00E4 | CJNE R7, #33H, 14H

00E7 | SJMP 0FH

00E9 | MOV 81H, 30H

00EC | SJMP 0DH

00EE | MOV 81H, 31H

00F1 | SJMP 08H

00F3 | MOV 81H, 32H

00F6 | SJMP 03H

00F8 | MOV 81H, 33H

00FB | POP 0D0H

00FD | POP 83H

00FF | POP 82H

0101 | POP 0F0H

0103 | POP 0E0H

0105 | RET

0106* | CLR 0AFH

0108 | MOV A, #0FH

010A | ANL A, 34H

010C | MOV R6, A

010D | MOV A, 35H

PC: 0x0106 | SP: 0x09

Modify RAM: addr 0x00 0x00 value

Data Memory:

0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	00	00	00	00	00	00	00	01	31	D0	00	00	00	33	01
10	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
20	03	30	41	4F	00	00	00	00	00	00	00	00	00	00	00
30	3F	4F	5F	6F	00	20	00	00	00	00	00	31	00	00	00
40	A4	00	00	00	00	00	00	00	00	00	00	00	00	00	00
50	14	00	00	00	00	08	00	00	00	00	00	00	00	00	00
60	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
70	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

Copyright © 2005-2024 James Rogers | Remove All Breakpoints

DI | LD

AND Gate Disabled

Key Bounce Disabled

Standard

U No Parity 8-bit UART @ 4800 Baud

Rx Reset

Tx Send

0.0 V output

Scope

DAC

BF 0 AC 0x00 IR 0x00 DR 0x00

0.0 V input

11111111

ADC

MAX

MIN

Motor Enabled

Here, this is moments before ThreadCreate(main) was called, I can know that because the PC is 0x0106 and it is the same with the data in the map file where the ThreadCreate first starts. I can also see that the SP at that time point is 0x09 which is different from the starting 0x07.

EdSim51DI - Version 2.1.38 & Dynamic Interface x | testpreempt.hex

System Clock (MHz): 11.0592 | Update Freq. 1

SBUF

R/O	W/O	TH0	TL0	R7	0x00	B	0x00
0x00	0x00	0x01	0x00	R6	0x00	ACC	0x00
RxD	TxD	TMOD	0x00	R5	0x00	PSW	0x80
1	1	TCON	0x10	R4	0x00	IP	0x00
SCON	0x00	PCON	0x00	R3	0x00	IE	0x02
				R2	0x00	DPH	0x00
				R1	0x00	DPL	0x56
				R0	0x00	SP	0x41

pins bits TH1 TL1

0xFF	0xFF	P3	0x00	0x00
0xFF	0xFF	P2	0x00	0x00
0xFF	0xFF	P1	0x00	0x00
0xFF	0xFF	P0	0x00	0x00

PC: 0x016B | SP: 0x41

8051

Data Memory

addr	0x00	0x00	value
0	00	00	00
1	00	00	00
2	00	00	00
3	00	00	00
4	00	00	00
5	00	00	00
6	00	00	00
7	00	00	00
8	00	00	00
9	00	00	00
A	00	00	00
B	00	00	00
C	00	00	00
D	00	00	00
E	00	00	00
F	00	00	00

Modify RAM

addr: 0x00 | value: 0x00

Remove All Breakpoints

Copyright ©2005-2024 James Rogers

RST Step Run New Load Save CPY Paste BP

Time: 55us - Instructions: 32

```

0138 | MOV A, 35H
013A | MOV 23H, 31H
013D | SJMP 22H
013F | MOV A, 34H
0141 | JB 0E3H, 0DH
0144 | MOV 3CH, #32H
0147 | ORL 34H, #04H
014A | MOV A, 35H
014C | MOV 23H, 32H
014F | SJMP 10H
0151 | MOV A, 34H
0153 | JB 0E3H, 0BH
0156 | MOV 3CH, #33H
0159 | ORL 34H, #08H
015C | MOV A, 35H
015E | MOV 23H, 33H
0161 | MOV 22H, 81H
0164 | MOV 81H, 23H
0167 | PUSH 82H
0169 | PUSH 83H
016B | MOV A, 00H

```

P0.7 1 Display-select Decoder CS|DAC WR
P0.6 1 Keypad Column 2
P0.5 1 Keypad Column 1
P0.4 1 Keypad Column 0
P0.3 1 Keypad Row 3
P0.2 1 Keypad Row 2
P0.1 1 Keypad Row 1
P0.0 1 Keypad Row 0
P1.7 1 LED 7|Seg. dp|DAC DB7|LCD DB7
P1.6 1 LED 6|Seg. g|DAC DB6|LCD DB6
P1.5 1 LED 5|Seg. f|DAC DB5|LCD DB5
P1.4 1 LED 4|Seg. e|DAC DB4|LCD DB4
P1.3 1 LED 3|... d|...DB3|...DB3|... RS
P1.2 1 LED 2|... c|...DB2|...DB2|LCD E
P1.1 1 LED 1|Seg. b|DAC DB1|LCD DB1
P1.0 1 LED 0|Seg. a|DAC DB0|LCD DB0
P2.7 1 SW 7|ADC DB7
P2.6 1 SW 6|ADC DB6
P2.5 1 SW 5|ADC DB5
P2.4 1 SW 4|ADC DB4
P2.3 1 SW 3|ADC DB3
P2.2 1 SW 2|ADC DB2
P2.1 1 SW 1|ADC DB1
P2.0 1 SW 0|ADC DB0
P3.7 1 ADC RD|Comparator Output
P3.6 1 ADC WR
P3.5 1 Motor Sensor
P3.4 1 Display-select Input 1
P3.3 1 AND Gate Output|Display-se..t 0
P3.2 1 ADC INTR
P3.1 1 Motor Control Bit 1|Ext. UART Rx
P3.0 1 Motor Control Bit 0|Ext. UART Tx

Here, this is before the second ThreadCreate was called, the SP is now in the range of 40-4F which means that we are now in the main thread.

EdSim51DI - Version 2.1.38 & Dynamic Interface x | testpreempt.hex

System Clock (MHz): 11.0592 | Update Freq. 1

SBUF

R/O	W/O	TH0	TL0	R7	0x30	B	0x00
0x00	0x00	0x04	0x15	R6	0x02	ACC	0x00
RxD	TxD	TMOD	0x00	R5	0x31	PSW	0x08
1	1	TCON	0x10	R4	0x00	IP	0x00
SCON	0x00	PCON	0x00	R3	0x00	IE	0x02
				R2	0x00	DPH	0x00
				R1	0x00	DPL	0x14
				R0	0xD0	SP	0x56

pins bits TH1 TL1

0xFF	0xFF	P3	0x00	0x00
0xFF	0xFF	P2	0x00	0x00
0xFF	0xFF	P1	0x00	0x00
0xFF	0xFF	P0	0x00	0x00

PC: 0x019A | SP: 0x56

8051

Data Memory

addr	0x00	0x00	value
0	00	00	00
1	00	00	00
2	00	00	00
3	00	00	00
4	00	00	00
5	00	00	00
6	00	00	00
7	00	00	00
8	00	00	00
9	00	00	00
A	00	00	00
B	00	00	00
C	00	00	00
D	00	00	00
E	00	00	00
F	00	00	00

Modify RAM

addr: 0x00 | value: 0x00

Remove All Breakpoints

Copyright ©2005-2024 James Rogers

RST Step Pause New Load Save CPY Paste BP

Time: 180us - Instructions: 97

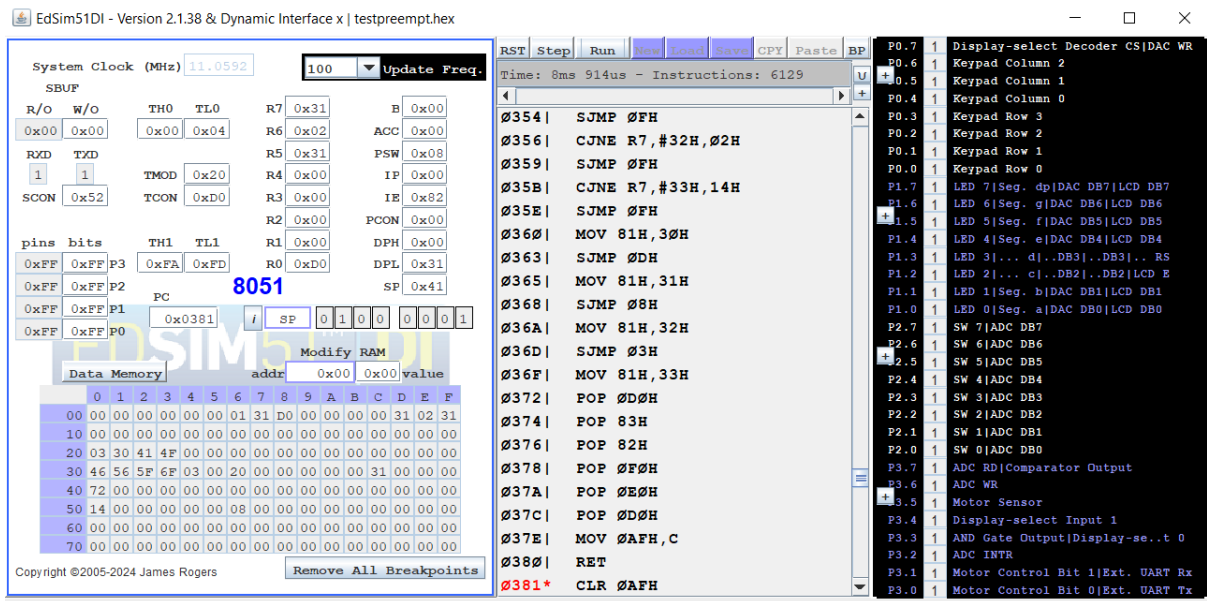
```

ORG 0000H
0000 | LJMP 00A7H
0003 | RETI
ORG 000BH
000B | LJMP 00AEH
000E | LJMP 0096H
0011 | LJMP 000EH
0014 | SETB 00H
0016 | JBC 0AFH, 02H
0019 | CLR 00H
001B | MOV 36H, #40H
001E | MOV C, 00H
0020 | MOV 0AFH, C
0022 | MOV A, 37H
0024 | ORL A, 38H
0026 | JNZ 0FAH
0028 | SETB 00H
002A | JBC 0AFH, 02H
002D | CLR 00H
002F | MOV A, #01H
0031 | ADD A, 37H

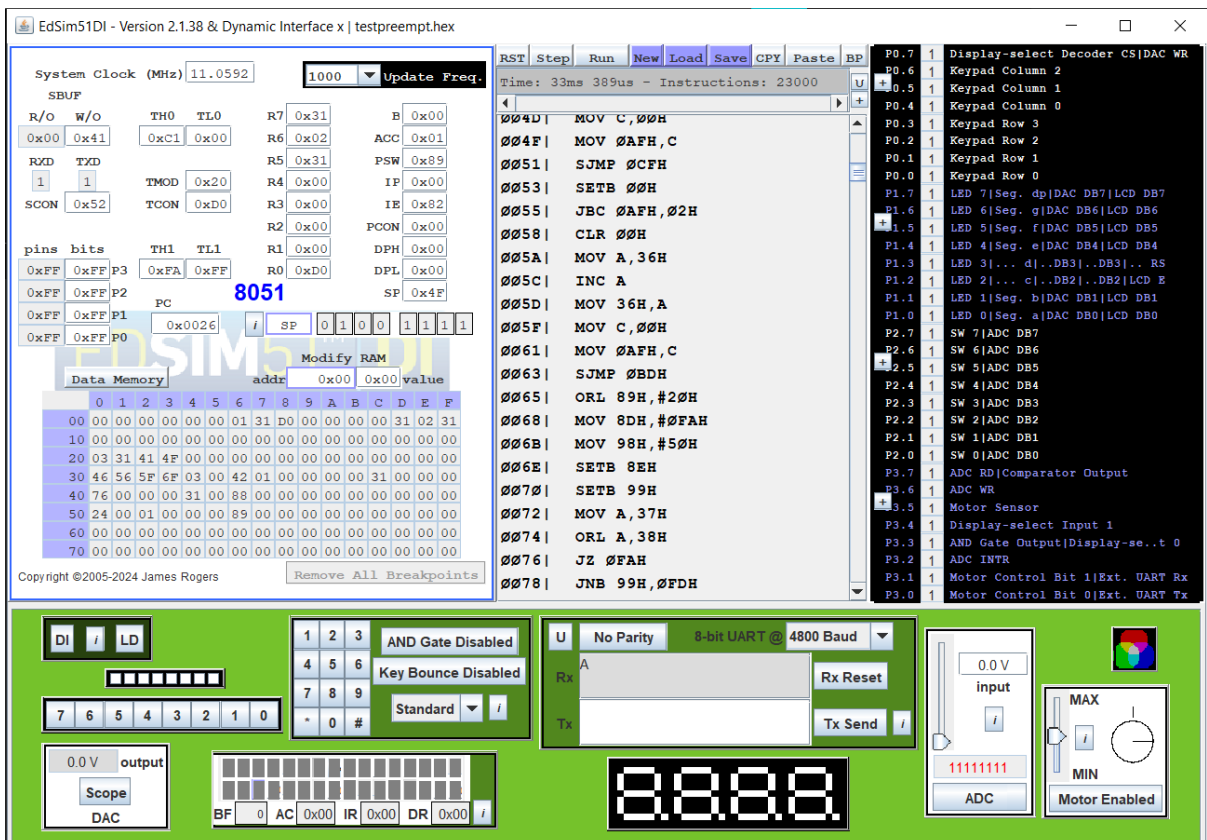
```

P0.7 1 Display-select Decoder CS|DAC WR
P0.6 1 Keypad Column 2
P0.5 1 Keypad Column 1
P0.4 1 Keypad Column 0
P0.3 1 Keypad Row 3
P0.2 1 Keypad Row 2
P0.1 1 Keypad Row 1
P0.0 1 Keypad Row 0
P1.7 1 LED 7|Seg. dp|DAC DB7|LCD DB7
P1.6 1 LED 6|Seg. g|DAC DB6|LCD DB6
P1.5 1 LED 5|Seg. f|DAC DB5|LCD DB5
P1.4 1 LED 4|Seg. e|DAC DB4|LCD DB4
P1.3 1 LED 3|... d|...DB3|...DB3|... RS
P1.2 1 LED 2|... c|...DB2|...DB2|LCD E
P1.1 1 LED 1|Seg. b|DAC DB1|LCD DB1
P1.0 1 LED 0|Seg. a|DAC DB0|LCD DB0
P2.7 1 SW 7|ADC DB7
P2.6 1 SW 6|ADC DB6
P2.5 1 SW 5|ADC DB5
P2.4 1 SW 4|ADC DB4
P2.3 1 SW 3|ADC DB3
P2.2 1 SW 2|ADC DB2
P2.1 1 SW 1|ADC DB1
P2.0 1 SW 0|ADC DB0
P3.7 1 ADC RD|Comparator Output
P3.6 1 ADC WR
P3.5 1 Motor Sensor
P3.4 1 Display-select Input 1
P3.3 1 AND Gate Output|Display-se..t 0
P3.2 1 ADC INTR
P3.1 1 Motor Control Bit 1|Ext. UART Rx
P3.0 1 Motor Control Bit 0|Ext. UART Tx

This is the time before the MyTimer0Handler is called because here the SP is in the range of 50-5F which means that the thread that is running is the producer.



This is the time before the MyTimer0Handler is called because we can see the range of the SP is 40-4F right now and that means the consumer is now running.



Above you can see that it finally printed out “A” that means the consumer works and it runs.

We can know that the interrupt will trigger on a regular basis because when the Timer0 overflows, interrupt happens and reset the timer back to 0.

