ECE/CS 3720: Embedded System Design (ECE 6960/2 and CS 6968)

Chris J. Myers
Lecture 11: Threads

Introduction to Threads

- Interrupts create a multithreaded environment with a single foreground thread (the main program), and multiple background threads (the ISRs).
- Projects where modules are loosely coupled, multiple foreground threads may be necessary.
- The chapter presents techniques to implement multiple foreground threads (the scheduler).
- It also presents synchronization tools, semaphores, that allow threads to interact with each other.
Thread Control Block

- A thread control block (TCB) stores information private to each thread, and it must contain:
  1. A pointer so that it can be chained into a linked list.
  2. The value of its stack pointer.
  3. A stack area for local variables and saved registers.
- A TCB may also contain:
  1. Thread number, type, or name.
  2. Age, or how long this thread has been active.
  3. Priority.
  4. Resources that this thread has been granted.

Thread Registers

**TCB of a running thread**
- TCB link
- stack pointer
- Id
- stack area
- local variables
- return pointers

**TCB of a thread not running**
- TCB link
- stack pointer
- Id
- stack area
- local variables
- return pointers
Assembly for the Threads

**ProxA pshx**
- tsx
- ldd #5
- std 0,x
- LoopA: ldaa #3
- staa PORTC
- ldd 0,x
- jsr sub
- std 0,x
- bra LoopA

**ProxB pshx**
- tsx
- ldd #6
- std 0,x
- LoopB: ldaa #4
- staa PORTC
- ldd 0,x
- jsr sub
- std 0,x
- bra LoopB

**Sub pshx**
- tsx
- std 0,x
- ldd #1
- Isa
- std 0,x
- prlx
- rts

Thread Control Block in Assembly

<table>
<thead>
<tr>
<th>TCB1 fdb</th>
<th>TCB2 fdb</th>
<th>TCB3 fdb</th>
<th>TCB1 link</th>
</tr>
</thead>
<tbody>
<tr>
<td>fdb IS1</td>
<td>fdb IS2</td>
<td>fdb IS3</td>
<td>SP</td>
</tr>
<tr>
<td>fcb 1</td>
<td>fcb 2</td>
<td>fcb 4</td>
<td>Id</td>
</tr>
<tr>
<td>rmb 49</td>
<td>rmb 49</td>
<td>rmb 49</td>
<td></td>
</tr>
<tr>
<td>IS1 rmb 1</td>
<td>IS2 rmb 1</td>
<td>IS3 rmb 1</td>
<td></td>
</tr>
<tr>
<td>fcb $40</td>
<td>fcb $40</td>
<td>fcb $40</td>
<td>CCR</td>
</tr>
<tr>
<td>fdb 0,0,0</td>
<td>fdb 0,0,0</td>
<td>fdb 0,0,0</td>
<td>DXY</td>
</tr>
<tr>
<td>fdb ProxA</td>
<td>fdb ProxA</td>
<td>fdb ProxB</td>
<td>PC</td>
</tr>
</tbody>
</table>

Thread Control Block in C

```c
struct TCB{
    struct TCB *Next; /* Link to Next TCB */
    unsigned char *SP; /* SP when not running */
    unsigned int Id; /* output to PortB */
    unsigned char MoreStack[49]; /* more stack */
    unsigned char CCR; /* Initial CCR */
    unsigned char RegB; /* Initial RegB */
    unsigned char RegA; /* Initial RegA */
    unsigned int RegX; /* Initial RegX */
    unsigned int RegY; /* Initial RegY */
    void (*PC)(void); /* Initial PC */
};
typedef struct TCB TCBType;
typedef TCBType *TCBPtr;
```

C for the Threads

```c
int Sub(int j){ int i;
    PORTC=1; /* Port C=program being executed */
    i=j+1;
    return(i);}
void ProxA(){ int i;
    i=5;
    while(1) { PORTC=2; i=Sub(i);}}
void ProxB(){ int i;
    i=6;
    while(1) { PORTC=4; i=Sub(i);}}
```
Thread Control Block in C

```
TCBType sys[3]={(  
  { &sys[1], /* Pointer to Next */  
    &sys[0].MoreStack[49], /* Initial SP */  
    1, { 0 }, /* Id, clear stack */  
    0x40,0,0,0,0, /* CCR,B,A,X,Y */  
    ProgA, }, /* Initial PC */  
  { &sys[2], /* Pointer to Next */  
    &sys[1].MoreStack[49], /* Initial SP */  
    2, { 0 }, /* Id, clear stack */  
    0x40,0,0,0,0, /* CCR,B,A,X,Y */  
    ProgA, }, /* Initial PC */  
  { &sys[0], /* Pointer to Next */  
    &sys[2].MoreStack[49], /* Initial SP */  
    4, { 0 }, /* Id, clear stack */  
    0x40,0,0,0,0, /* CCR,B,A,X,Y */  
    ProgB, } }; /* Initial PC */
```

Preemptive Thread Scheduler in Assembly (cont)

```
* launch next thread
  ldx Next,x
Start stx RunPt
  ldaa Id,x
  staa PORTB  // visualizes running thread
  lds SP,x  // set SP for this new thread
  ldd TOC5
  addd #20000  // interrupts every 10 ms
  std TOC5
  ldaa #$08  // ($20 on the 6812)
  staa TFLG1  // acknowledge OC5
  rti
```

Preemptive Thread Scheduler in Assembly

```
Next equ 0  // pointer to next TCB
SP equ 2  // Stack pointer for this thread
Id equ 4  // Used to visualize thread running
RunPt mb 2  // pointer to thread running
Main ldaa #$FF
  staa DDRC  // PortC displays program executing
  ldx #TCB1  // First thread to run
  jmp Start
  * Suspend thread which is currently running
  OC5Han lda RunPt
  sts SP,x  // save Stack Pointer in TCB
```

Preemptive Thread Scheduler in C

```
TCBPtr RunPt;  // Pointer to current thread  */
#pragma interrupt_handler ThreadSwitch()
void ThreadSwitch(){
  asm("ldx _RunPt\n"
    "sta PORTB\n"
    "ldaa PORTB\n"
    "staa PORTB\n"
    "LDX #TCB1\n"
    "JMP Start\n"
    "SUSPEND\n"
    "OC5Han lda RunPt\n"
    "sts SP,x\n"
    /* TCBPtr RunPt; */
  )
```
Preemptive Thread Scheduler in C (cont)

```c
void main(void){
    DDRC=0xFF; /* Specify first thread */
    asm(" sei");
    TOC3vector=&ThreadSwitch; TFLG1 = 0x20; /* Clear OC3F */
    TMSK1 = 0x20; /* Arm TOC3 */
    TOC3=TCNT+20000;
    PORTB=RunPt->Id;
    asm(" ldx _RunPt\n" 
" lds 2,x\n" 
" cli\n"
" rti"); /* Launch First Thread */
```

Other Scheduling Algorithms

- A non-preemptive (cooperative) scheduler trusts each thread to voluntarily release control on a periodic basis.
- Not appropriate for real-time systems.
- A priority scheduler assigns a priority to each thread.
- A thread is scheduled only if no higher priority thread is ready.
- Priority reduces latency for important tasks.
- In a busy system, low-priority threads may starve.

Dynamic Allocation of Threads

```c
void create(void (*program)(void), int TheId){
    TCBPtr NewPt; // pointer to new TCB
    NewPt=(TCBPtr)malloc(sizeof(TCBType));
    if(NewPt==0)return;
    NewPt->Id=TheId; /* used to visualize */
    NewPt->CCR=0x40; /* InitialCCR, I=0 */
    NewPt->RegB=0; /* Initial RegB */
    NewPt->RegA=0; /* Initial RegA */
    NewPt->RegX=0; /* Initial RegX */
    NewPt->RegY=0; /* Initial RegY */
    NewPt->PC=program; /* Initial PC */
    if(RunPt){
        NewPt->Next=RunPt->Next;
        RunPt->Next=NewPt; /* will run Next */
    }
    else
        RunPt=NewPt; /* first & only thread */
```