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
C for the Threads

```c
int Sub(int j){ int i;
    PORTC=1; /* Port C=program being executed */
    i=j+1;
    return(i);}
void ProgA(){ int i;
    i=5;
    while(1) { PORTC=2; i=Sub(i);}}
void ProgB(){ int i;
    i=6;
    while(1) { PORTC=4; i=Sub(i);}}
```

Assembly for the Threads

```
ProgA pshx
    tsx
    ldd #5
    std 0,x
LoopA ldaa #2
    staa PORTC
    ldd 0,x
    jsr sub
    std 0,x
bra LoopA

ProgB 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
    ldaa #1
    pulx
    rts
```

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;
```

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 C

Preemptive Thread Scheduler in Assembly

Preemptive Thread Scheduler in C (cont)
Preemptive Thread Scheduler in Assembly (cont)

* launch next thread
  
  idx Next, x

Start

  stx RunPt
  ldaa Id, x
  staa PORTB visualizes running thread
  lds SP, x set SP for this new thread

ldd TOC5

add #20000 interrupts every 10 ms
std TOC5

ldaa #$08 ($20 on the 6812)
staa TFLG1 acknowledge OC5
rti

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

void create(void (*program)(void), int TheId){
  TCBPtr NewPt; // pointer to new TCB
  NewPt=(TCBPtr)malloc(sizeof(TCBType));
  if(NewPt==0)return;
  NewPt->SP=&(NewPt->CCR-1); /*SP when not running */
  NewPt->Id=TheId; /* used to visualize */
  NewPt->CCR=0x40; /* Initial CCR, I=0 */
  NewPt->RegA=0; /* Initial RegA */
  NewPt->RegB=0; /* Initial RegB */
  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 */