Lecture 11: Threads
We start by looking at the implementation side
Later, we look at how to use threads
Interrupts create a concurrent environment with a single foreground thread (the main program) plus the ISRs. In projects where modules are loosely coupled, multiple foreground threads may be necessary.
Why use threads?

Can improve program responsiveness.
Can improve program modularity through decoupling.
Blocking is very convenient for programmers.
Thread Memory

Thread1

A | B
X
Y
SP
PC

Stack

Thread2

A | B
X
Y
SP
PC

Stack

Thread3

A | B
X
Y
SP
PC

Stack

Thread1

pt

Thread2

pt

Thread3

pt

Global
State Machine for a Single Thread

Create thread

Active

Resource available

OS grants control

Time slice over, OS takes control away

Blocked

Run

Thread needs resource

Kill thread
Thread Lists

```
Thread1 → Next → RunPt → Next → Thread5 → Next → Thread2
```

```
BlockOnPrinterPt → Next → Thread6
```

```
BlockOnEmptyPt → Next → Thread4 → Next → Thread3
```

```
BlockOnFullPt = Null
```
Scheduling is the process by which the system determines which thread to run next.

Scheduling decisions happen when threads change state (Run → Blocked, etc.) or are created or deleted.

Nonpreemptive scheduling is when the scheduler only chooses a new thread/process after the current one terminates or blocks.

Preemptive scheduling is when the scheduler may choose a new thread/process even though the current one is Active. Preemptive schedulers can result in more responsive systems but require more effort from the programmer to create a correct system.
Scheduling Metrics

- CPU utilization.
- Throughput.
- Turnaround time.
- Waiting time.
- Response time.
Scheduler Types

First-Come, First-Served.
Shortest-Job-First.
Priority.
Round-Robin.
Multi-level Queue and Variants.
Round-Robin Scheduler
A thread control block (TCB) stores information for managing each thread, and it must contain:

- A pointer so that it can be chained into a linked list.
- The value of its stack pointer.
- A stack area for local variables and saved registers.

A TCB may also contain:

- Thread number, type, or name.
- Age, or how long this thread has been active.
- Priority.
- Resources that this thread has been granted.
Thread Registers

TCB of a running thread

- 6811
- CC, B, A
- X, Y, PC
- SP
- TCB link
- Stack pointer
- Id
- Stack area
- Local variables
- Return pointers

TCB of a thread not running

- TCB link
- Stack pointer
- Id
- Stack area
- CC, B, A
- X, Y, PC
- Local variables
- Return pointers
int Sub(int j) { int i;
    PTM = 1;  // Port M
    i = j+1;
    return(i); }
void ProgA() { int i;
    i=5;
    while(1) {
        PTM = 2;
        i = Sub(i); }
void ProgB() { int i;
    i=6;
    while(1) {
        PTM = 4;
        i = Sub(i); }
Thread Control Block in C

```c
struct TCB
{
    struct TCB *Next;    /* Link to Next TCB */
    unsigned char *SP;   /* Stack Pointer when idle */
    unsigned short Id;  /* output to PortT */
    unsigned char MoreStack[49];  /* more stack */
    unsigned char CCR;   /* Initial CCR */
    unsigned char RegB;  /* Initial RegB */
    unsigned char RegA;  /* Initial RegA */
    unsigned short RegX; /* Initial RegX */
    unsigned short RegY; /* Initial RegY */
    void (*PC)(void);    /* Initial PC */
};

typedef struct TCB TCBType;
typedef TCBType * TCBPtr;
```
TCBType sys[3]=
{
    &sys[1],            /* Pointer to Next */
    &sys[0].CCR,        /* Initial SP */
    1,                 /* Id */
    { 0},
    0x40,0,0,0,0,       /* CCR,B,A,X,Y */
    ProgA },            /* Initial PC */
{
    &sys[2],            /* Pointer to Next */
    &sys[1].CCR,        /* Initial SP */
    2,                 /* Id */
    { 0},
    0x40,0,0,0,0,       /* CCR,B,A,X,Y */
    ProgA },            /* Initial PC */
{
    &sys[0],            /* Pointer to Next */
    &sys[2].CCR,        /* Initial SP */
    4,                 /* Id */
    { 0},
    0x40,0,0,0,0,       /* CCR,B,A,X,Y */
    ProgB } };          /* Initial PC */
Preemptive Thread Scheduler in C

TCBPtr RunPt;    /* Pointer to current thread */
void main(void) { 
    DDRT = 0xFF;   /* Output running thread on Port T */
    DDRM = 0xFF;   /* Output running program on Port M */
    RunPt = &sys[0];  /* Specify first thread */
    asm sei
    TFLG1 = 0x20;   /* Clear C5F */
    TIE = 0x20;    /* Arm C5F */
    TSCR1 = 0x80;  /* Enable TCNT*/
    TSCR2 = 0x01;  /* 2MHz TCNT */
    TIOS |= 0x20; /* Output compare */
    TC5 = TCNT+20000;
    PTT = RunPt->Id;
    asm ldx RunPt
    asm lds 2,x
    asm cli
    asm rti
}   /* Launch First Thread */
void interrupt 13 ThreadSwitch() {
    asm ldx RunPt
    asm sts 2,x
    RunPt = RunPt->Next;
    PTT = RunPt->Id; /* PortH=active thread */
    asm ldx RunPt
    asm lds 2,x
    TC5 = TCNT+20000; /* Thread runs for 10 ms */
    TFLG1 = 0x20; } /* ack by clearing C5F */
int create(void (*startFunc)(void), int TheId) {
    TCBPtr NewPt;    // pointer to new thread control block
    NewPt = (TCBPtr)malloc(sizeof(TCBType)); // new TCB
    if (NewPt == 0) return FAIL;
    NewPt->SP = &(NewPt->CCR); /* Stack Pointer when not running */
    NewPt->Id = TheId; /* Visualize active thread */
    NewPt->CCR = 0x40; /* Initial CCR, I=0 */
    NewPt->RegB = 0; /* Initial RegB */
    NewPt->RegA = 0; /* Initial RegA */
    NewPt->RegX = 0; /* Initial RegX */
    NewPt->RegY = 0; /* Initial RegY */
    NewPt->PC = startFunc; /* Initial PC */
    if (RunPt) {
        NewPt->Next = RunPt->Next;
        RunPt->Next = NewPt; /* will run Next */
    } else
        RunPt = NewPt; /* the first and only thread */
    return SUCCESS;
}
This was the implementation side of a very simple thread system
It is not that hard!
A preemptive threading system is the core of an RTOS
Designing correct embedded code that uses threads is the hard part