ECE/CS 5780/6780: Embedded System Design

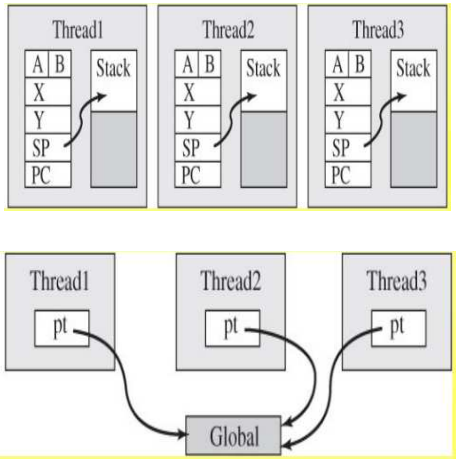
Chris J. Myers

Lecture 10: Threads

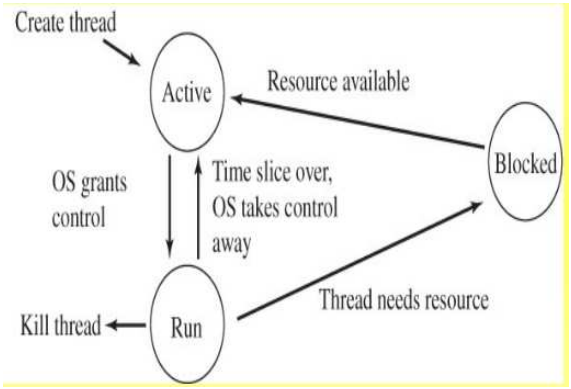
Introduction to Threads

- 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.
- This chapter presents techniques to implement multiple foreground threads (the *scheduler*).
- It also presents synchronization tools, *semaphores*, that allow threads to interact with each other.

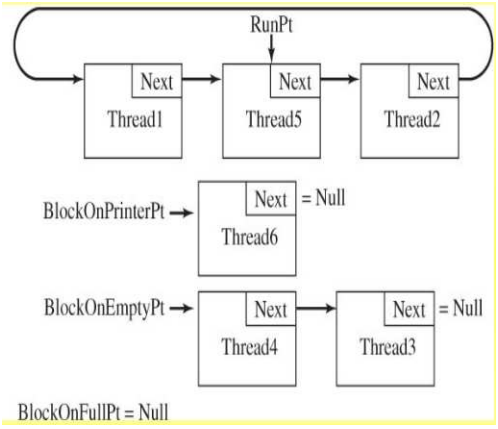
Thread Memory



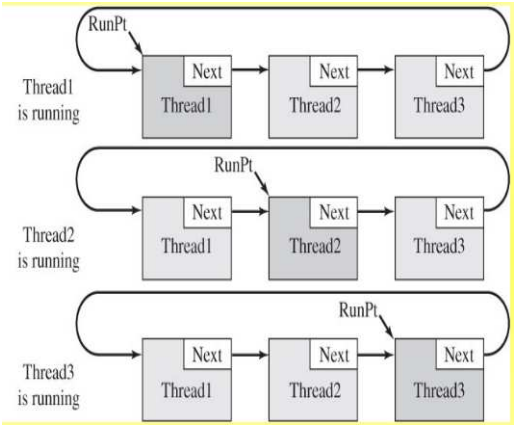
Thread States



Thread Lists



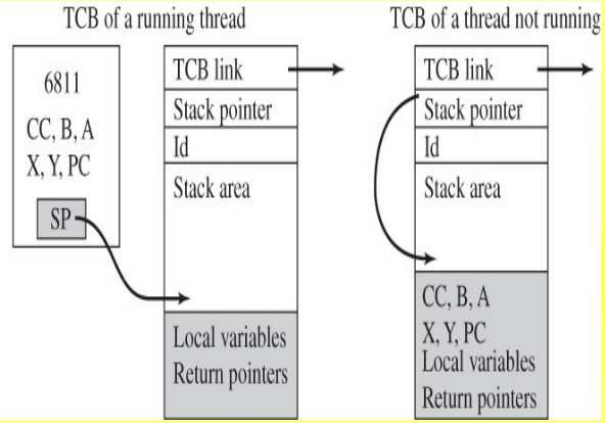
Round-Robin Scheduler



Thread Control Block

- A *thread control block* (TCB) stores information private to 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



C for the Threads

```
int Sub(int j){ int i;
  PTM = 1;  // PTM=program being executed
  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

```
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;
```


Thread Control Block in C

```
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 */
```

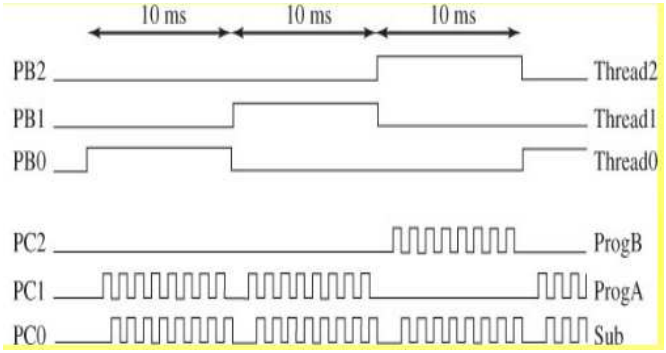

Preemptive Thread Scheduler in C

```
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 */
```

Preemptive Thread Scheduler in C (cont)

```
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 */
```


Profile of Three Threads



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 thread control block
  NewPt = (TCBPtr)malloc(sizeof(TCBType)); // new TCB
  if(NewPt==0)return;
  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=program;          /* Initial PC */
  if(RunPt){
    NewPt->Next = RunPt->Next;
    RunPt->Next = NewPt;}     /* will run Next */
  else
    RunPt = NewPt;}           /* the first and only thread */
```