

## 5. Threads

### 5.1. Multithreaded Preemptive Scheduler

#### how to create a new thread

- allocate TCB
- link TCB into ready queue, set **Next** field
- set **Id** field
- initial CC,B,A,X,Y, PC in stack area (like was interrupted)
- no local variables
- initial SP in **StackPt** field

```

struct    TCB{
    struct TCB *Next;      // Link to Next TCB
    unsigned char *StackPt;
    unsigned char Id;      // 1, 2, 4, ...
    unsigned char MoreStack[100];
    unsigned char InitialCCR;
    unsigned char InitialRegB;
    unsigned char InitialRegA;
    unsigned short InitialRegX;
    unsigned short InitialRegY;
    void (*InitialPC)(void);
};
    
```

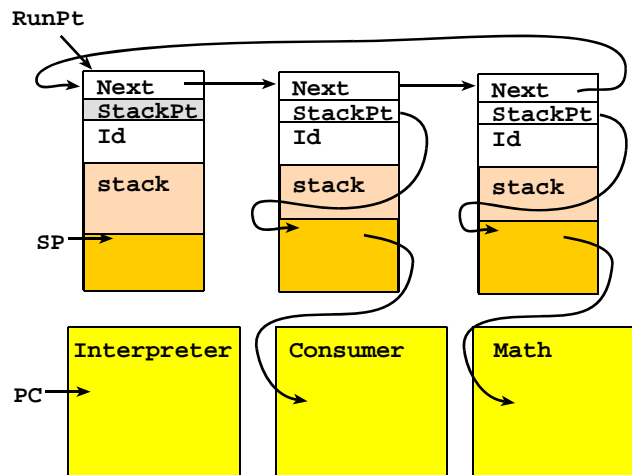


Figure 5.5. The circular linked list allows the scheduler to run all three threads equally.

- Open Lab17.c in Metrowerks
- Show actual **OS\_AddThread**
- Open Lab17.uc in TExaS
- Run, and observe **TCB**

## 5.2. Semaphores

P or wait (derived from the Dutch word *proberen*  
 V or signal (derived from the Dutch word *verhogen*

meaning or significance to the counter value.  
 binary semaphore a global variable that can be  
 1 for free and  
 0 for busy.

### 5.2.1. Spin-lock semaphore implementation

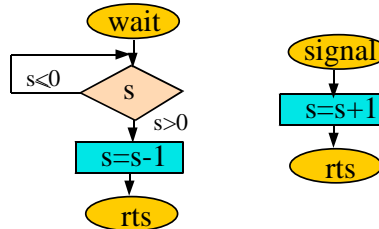


Figure 5.8. Flowcharts of a spinlock counting semaphore.

**Observation:** If the semaphores can be implemented without disabling interrupts, then the latency in response to external events will be improved.

Show C code for signal and wait  
 Show applications of semaphore

## 5.3. Applications of semaphores

### 5.3.2. Resource sharing, or mutual exclusion

| Interpreter                            | Consumer                               |
|--|--|
| <code>Wait(&amp;DisplayFree);</code>   | <code>Wait(&amp;DisplayFree);</code>   |
| <code>OutString("bye");</code>         | <code>OutString("tchau");</code>       |
| <code>Signal(&amp;DisplayFree);</code> | <code>Signal(&amp;DisplayFree);</code> |