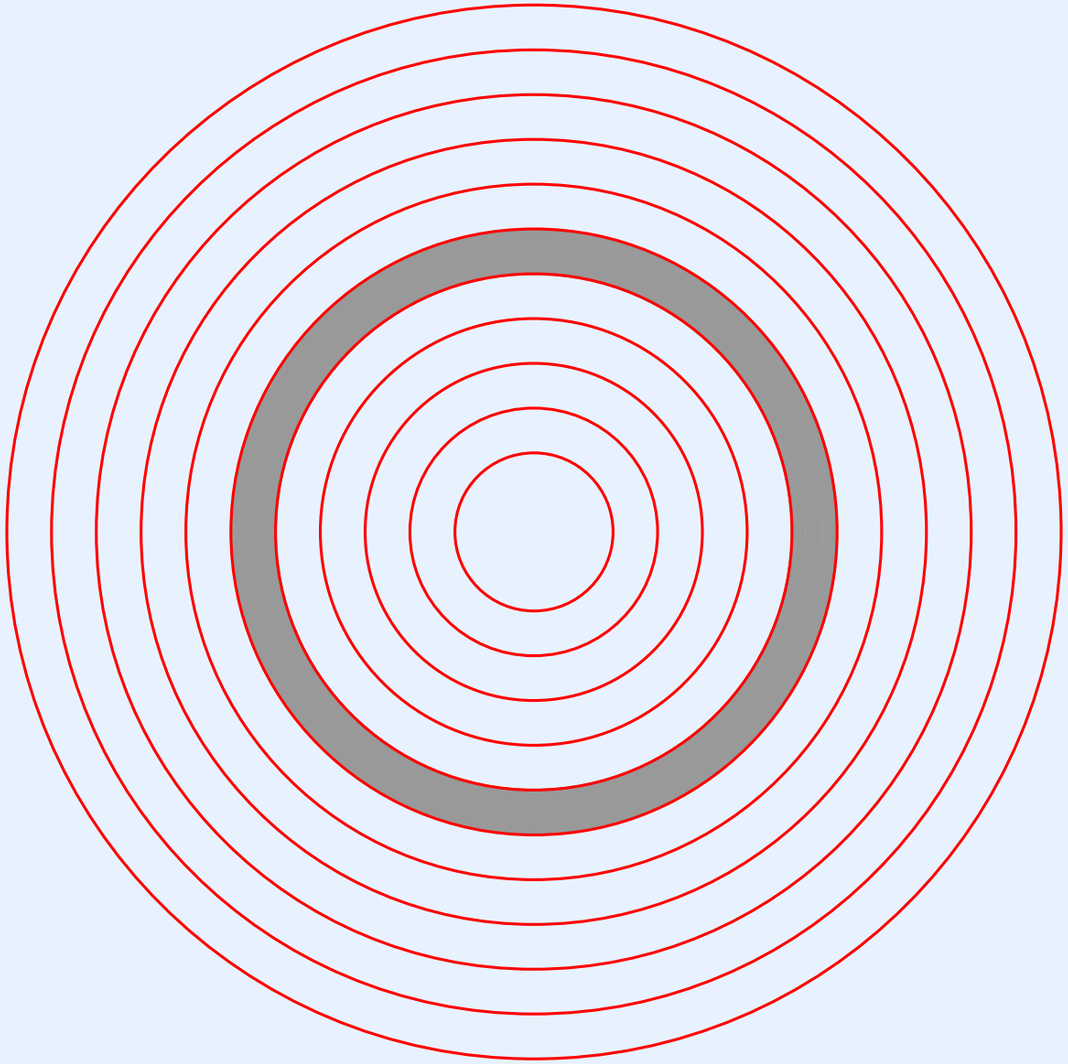
Module XIII

Real-Time Clock Management And Timed Events

Location Of Clock Management In The Hierarchy



Various Types Of Clock Hardware Exist

- Processor clock (rate at which instructions execute)
- Real-time clock
  - Pulses regularly
  - Interrupts the processor on each pulse
  - Called *programmable* if rate can be controlled by OS
- Interval timer
  - The processor sets a timeout and the device interrupts after the specified time
  - Can be used to pulse regularly
  - May have an automatic restart capability

Timed Events

- Two types of timed events are important to an operating system
- A *preemption event*
  - Known as *timeslicing*
  - Guarantees that a given process cannot run forever
  - Switches the processor to another process
- A *sleep event*
  - Is requested by a process to delay for a specified time
  - The process resumes execution after the time passes

A Note About Timeslicing

Most applications are I/O bound, which means the application is likely to perform an operation that takes the process out of the current state before its timeslice expires.

Managing Timed Events

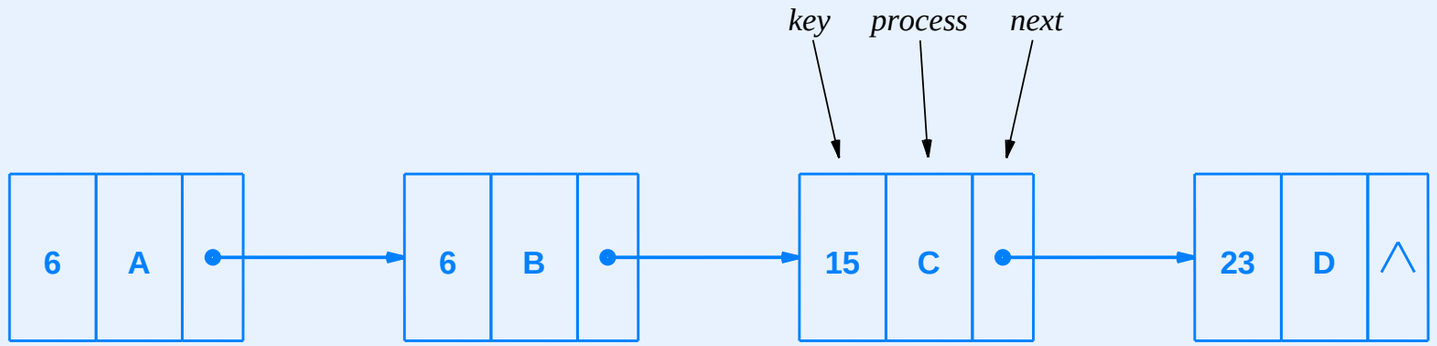
- The code must be efficient because
  - Clock interrupts occur frequently and continuously
  - More than one event may occur at a given time
  - The clock interrupt code must avoid searching a list of pending events
- An efficient mechanism
  - Keep all timed events on a list
  - Call the list an *event queue*

The Delta List

- A data structure used for timed events
- Items on a delta list are ordered by the time they will occur
- Trick to make processing efficient: use *relative* times
- Implementation: the key in an item stores the difference (*delta*) between the time for the event and time for the previous event
- The key in first event stores the delta from “now”

Delta List Example

- Assume events for processes *A* through *D* will occur *6*, *12*, *27*, and *50* ticks from now
- The delta keys are *6*, *6*, *15*, and *23*



Real-time Clock Processing In Xinu

- The clock interrupt handler
  - Decrements the preemption counter and calls *resched* if the timeslice has expired
  - Processes the sleep queue
- The sleep queue
  - Is a delta list
  - Each item on the list is a sleeping process
- Global variable *sleepq* contains the ID of the sleep queue

Keys On The Xinu Sleep Queue

- Processes on *sleepq* are ordered by time at which they will awaken
- Each key tells the number of clock ticks that the process must delay beyond the preceding one on the list
- The relationship must be maintained whenever an item is inserted or deleted

Sleep Timer Resolution

- A process calls *sleep* to delay
- Question: what resolution should be used for sleep?
  - Humans typically think of delays in seconds or minutes
  - Some applications may need millisecond accuracy (or more, if available)
- The tradeoff: using a high resolution, such as microseconds, means long delays will overflow a 32-bit integer

Xinu Sleep Primitives

- Xinu offers a set of functions to accommodate a range of possible resolutions

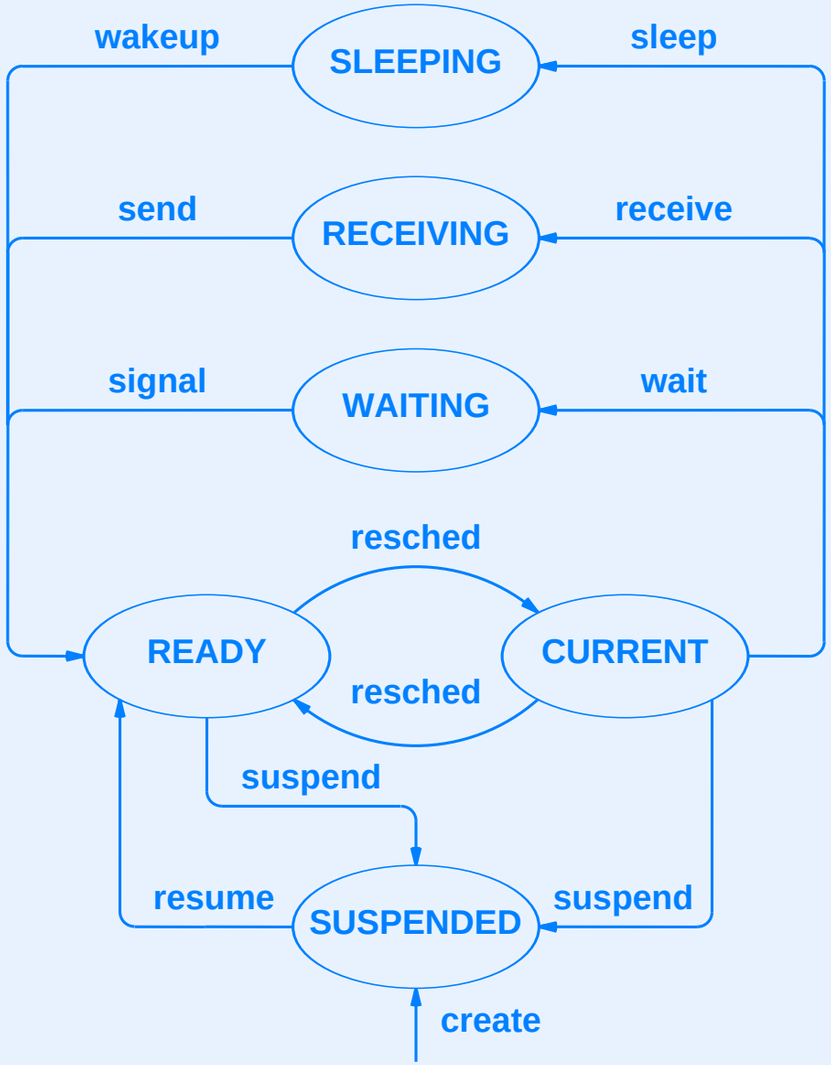
sleep	– the delay is given in seconds
sleep10	– the delay is given in tenths of seconds
sleep100	– the delay is given in hundredths of seconds
sleepms	– the delay is given in milliseconds

- The smallest resolution is milliseconds because the clock operates at a rate of one millisecond per tick

A New Process State For Sleeping Processes

- A sleeping process is not ready, suspended, or waiting
- A new state is required
  - The process enters the sleeping state by calling a sleep function
  - The clock interrupt handler calls *wakeup* when the delay expires

A New Process State For Sleeping Processes (continued)



Sleep.c (sleep and sleepms) (Part 1)

```
/* sleep.c - sleep sleepms */

#include <xinu.h>

#define MAXSECONDS      2147483         /* Max seconds per 32-bit msec  */

/*------------------------------------------------------------------------
 *  sleep  -  Delay the calling process n seconds
 *------------------------------------------------------------------------
 */
syscall sleep(
          int32 delay           /* Time to delay in seconds     */
        )
{
        if ( (delay < 0) || (delay > MAXSECONDS) ) {
                return SYSERR;
        }
        return sleepms(1000*delay);
}
```

Sleep.c (sleep and sleepms) (Part 2)

```
/*------------------------------------------------------------------------
 *  sleepms  -  Delay the calling process n milliseconds
 *------------------------------------------------------------------------
 */
syscall sleepms(
          int32 delay                   /* Time to delay in msec.       */
        )
{
        intmask mask;                   /* Saved interrupt mask         */

        if (delay < 0) {
                return SYSERR;
        }

        if (delay == 0) {
                yield();
                return OK;
        }
```

Sleep.c (sleep and sleepms) (Part 3)

```
        /* Delay calling process */

        mask = disable();
        if (insertd(currpid, sleepq, delay) == SYSERR) {
                restore(mask);
                return SYSERR;
        }

        proctab[currpid].prstate = PR_SLEEP;
        resched();
        restore(mask);
        return OK;
}
```

Inserting An Item On Sleepq

- The current process calls *sleepms* or *sleep* to request a delay
- *Sleepms*
  - The underlying function that takes action
  - Inserts current process on *sleepq*
  - Calls *resched* to allow other processes to execute
- Method
  - Walk through *sleepq* (with interrupts disabled)
  - Find the place to insert the process
  - Adjust remaining keys as necessary

Invariant For Insertion On sleepq

- The key of the first process on a delta list specifies the number of clock ticks a process must delay beyond the current time
- The key of each other process on a delta list specifies the number of clock ticks the process must delay beyond the preceding process on the list.
- When inserting a new delay on the list, the code adheres to the following invariant:

At any time during the search, both key **and** queuetab[next].qkey **specify a delay relative to the time at which the predecessor of the "next" process awakens.**

Xinu Insertd (Part 1)

```
/* insertd.c - insertd */

#include <xinu.h>

/*------------------------------------------------------------------------
 *  insertd  -  Insert a process in delta list using delay as the key
 *------------------------------------------------------------------------
 */
status  insertd(                        /* Assumes interrupts disabled  */
          pid32         pid,            /* ID of process to insert      */
          qid16         q,              /* ID of queue to use           */
          int32         key             /* Delay from "now" (in ms.)    */
        )
{
        int32   next;                   /* Runs through the delta list  */
        int32   prev;                   /* Follows next through the list*/

        if (isbadqid(q) || isbadpid(pid)) {
                return SYSERR;
        }
```

Xinu Insertd (Part 2)

```
        prev = queuehead(q);
        next = queuetab[queuehead(q)].qnext;
        while ((next != queuetail(q)) && (queuetab[next].qkey <= key)) {
                key -= queuetab[next].qkey;
                prev = next;
                next = queuetab[next].qnext;
        }

        /* Insert new node between prev and next nodes */

        queuetab[pid].qnext = next;
        queuetab[pid].qprev = prev;
        queuetab[pid].qkey = key;
        queuetab[prev].qnext = pid;
        queuetab[next].qprev = pid;
        if (next != queuetail(q)) {
                queuetab[next].qkey -= key;
        }

        return OK;
}
```

A Clock Interrupt Handler

- Updates the time-of-day (which counts seconds)
- Handles sleeping processes
  - Decrements the key of the first process on the sleep queue
  - Calls *wakeup* if the counter reaches zero
- Handles preemption
  - Decrements the preemption counter
  - Calls *resched* if the counter reaches zero

A Clock Interrupt Handler
(continued)

- When sleeping processes awaken
  - More than one process may awaken at a given time
  - The processes may not have the same priority
  - If the clock interrupt handler starts a process running immediately, a higher priority process may remain on the sleep queue, even if its time has expired
- Solution: *wakeup* awakens *all* processes that have zero time remaining before allowing any of them to run

Xinu Wakeup

```
/* wakeup.c - wakeup */

#include <xinu.h>

/*------------------------------------------------------------------------
 *  wakeup  -  Called by clock interrupt handler to awaken processes
 *------------------------------------------------------------------------
 */
void    wakeup(void)
{
        /* Awaken all processes that have no more time to sleep */

        resched_cntl(DEFER_START);
        while (nonempty(sleepq) && (firstkey(sleepq) <= 0)) {
                ready(dequeue(sleepq));
        }

        resched_cntl(DEFER_STOP);
        return;
}
```

- Note that rescheduling is deferred until all processes are awakened

Timed Message Reception

- Many operating system components offer a “timeout” on operations
- Timeout is especially useful in building communication protocols
- A Xinu example: receive with timeout
  - Operates like *receive*, but includes a timeout argument
  - If a message arrives before the timer expires, the message is returned
  - If the timer expires before a message arrives, the value *TIMEOUT* is returned
  - Implemented with *recvtime*
- Recvtime uses the same queue and wakeup mechanism as sleeping processes

Xinu Recvtime (Part 1)

```
/* recvtime.c - recvtime */

#include <xinu.h>

/*------------------------------------------------------------------------
 *  recvtime  -  Wait specified time to receive a message and return
 *------------------------------------------------------------------------
 */
umsg32  recvtime(
          int32         maxwait         /* Ticks to wait before timeout */

        )
{
        intmask mask;                   /* Saved interrupt mask         */
        struct  procent *prptr;         /* Tbl entry of current process */
        umsg32  msg;                    /* Message to return            */

        if (maxwait < 0) {
                return SYSERR;
        }
        mask = disable();
```

Xinu Recvtime (Part 2)

```
        /* Schedule wakeup and place process in timed-receive state */

        prptr = &proctab[currpid];
        if (prptr->prhasmsg == FALSE) { /* Delay if no message waiting  */
                if (insertd(currpid,sleepq,maxwait) == SYSERR) {
                        restore(mask);
                        return SYSERR;
                }
                prptr->prstate = PR_RECTIM;
                resched();
        }

        /* Either message arrived or timer expired */

        if (prptr->prhasmsg) {
                msg = prptr->prmsg;     /* Retrieve message             */
                prptr->prhasmsg = FALSE;/* Reset message indicator      */
        } else {
                msg = TIMEOUT;
        }
        restore(mask);
        return msg;
}
```

When A Process Sends A Message

- The target process could be in
  - The receiving state, PR_RECV
  - The receive-with-timeout state, PR_RECTIM
- A call to *send* handles both cases

Look Again At Send.c (Part 1)

```
/* send.c - send */

#include <xinu.h>

/*------------------------------------------------------------------------
 *  send  -  Pass a message to a process and start recipient if waiting
 *------------------------------------------------------------------------
 */
syscall send(
          pid32         pid,            /* ID of recipient process      */
          umsg32        msg             /* Contents of message          */
        )
{
        intmask mask;                   /* Saved interrupt mask         */
        struct  procent *prptr;         /* Ptr to process's table entry */

        mask = disable();
        if (isbadpid(pid)) {
                restore(mask);
                return SYSERR;
        }
```

Look Again At Send.c (Part 2)

```
        prptr->prmsg = msg;             /* Deliver message              */
        prptr->prhasmsg = TRUE;         /* Indicate message is waiting  */

        /* If recipient waiting or in timed-wait make it ready */

        if (prptr->prstate == PR_RECV) {
                ready(pid);
        } else if (prptr->prstate == PR_RECTIM) {
                unsleep(pid);
                ready(pid);
        }
        restore(mask);          /* Restore interrupts */
        return OK;
}
```

Unsleep - Remove A Sleeping Process (Part 1)

```
/* unsleep.c - unsleep */

#include <xinu.h>

/*------------------------------------------------------------------------
 *  unsleep  -  Internal function to remove a process from the sleep
 *                  queue prematurely.  The caller must adjust the delay
 *                  of successive processes.
 *------------------------------------------------------------------------
 */
status  unsleep(
          pid32         pid             /* ID of process to remove      */
        )
{
        intmask mask;                   /* Saved interrupt mask         */
        struct  procent *prptr;         /* Ptr to process's table entry */

        pid32   pidnext;                /* ID of process on sleep queue */
                                        /*   that follows the process   */
                                        /*   which is being removed     */

        mask = disable();
```

Unsleep - Remove A Sleeping Process (Part 2

```
        if (isbadpid(pid)) {
                restore(mask);
                return SYSERR;
        }

        /* Verify that candidate process is on the sleep queue */

        prptr = &proctab[pid];
        if ((prptr->prstate!=PR_SLEEP) && (prptr->prstate!=PR_RECTIM)) {
                restore(mask);
                return SYSERR;
        }

        /* Increment delay of next process if such a process exists */

        pidnext = queuetab[pid].qnext;
        if (pidnext < NPROC) {
                queuetab[pidnext].qkey += queuetab[pid].qkey;
        }

        getitem(pid);                   /* Unlink process from queue */
        restore(mask);
        return OK;
}
```

The Clock Hardware Interface

- The clock interface follows the pattern used by all devices
- The system uses a memory-mapped interaction
  - Some high bus addresses correspond to the clock device, not memory
  - When the processor stores data to one of the special addresses, the data being stored goes to the clock device
  - When the processor fetches from the special addresses, the clock device answers the request and sends information to the processor
  - Typically, the processor sends commands to a device
- A device driver defines a structure that specifies the layout of special addresses and their meaning as well as constants used (usually called *control and status registers*)

ARM Clock Definitions (Part 1)

```
/* clock.h */

extern  uint32  clktime;        /* current time in secs since boot      */
extern  uint32  count1000;      /* ms since last clock tick             */

extern  qid16   sleepq;         /* queue for sleeping processes         */
extern  int32   slnonempty;     /* nonzero if sleepq is nonempty        */
extern  int32   *sltop;         /* ptr to key in first item on sleepq   */
extern  uint32  preempt;        /* preemption counter                   */

struct am335x_timer1ms {
        uint32  tidr;           /* Identification register      */
        uint32  res1[3];        /* Reserved                     */
        uint32  tiocp_cfg;      /* OCP Interface register       */
        uint32  tistat;         /* Status register              */
        uint32  tisr;           /* Interrupt status register    */
        uint32  tier;           /* Interrupt enable register    */
        uint32  twer;           /* Wakeup enable register       */
        uint32  tclr;           /* Optional features            */
        uint32  tcrr;           /* Internal counter value       */
        uint32  tldr;           /* Timer load value             */
        uint32  ttgr;           /* Trigger register             */
        uint32  twps;           /* Write posting register       */
        uint32  tmar;           /* Match register               */
```

ARM Clock Definitions (Part 2)

```
        uint32  tcar1;          /* Capture register 1           */
        uint32  tsicr;          /* Synchronous interface control*/
        uint32  tcar2;          /* Capture register 2           */
        uint32  tpir;           /* Positive increment register  */
        uint32  tnir;           /* Negative increment register  */
        uint32  tcvr;           /* 1ms control register         */
        uint32  tocr;           /* Overflow mask register       */
        uint32  towr;           /* no. of overflows             */
};

#define AM335X_TIMER1MS_ADDR                    0x44E31000
#define AM335X_TIMER1MS_IRQ                     67

#define AM335X_TIMER1MS_TIOCP_CFG_SOFTRESET     0x00000002
#define AM335X_TIMER1MS_TISTAT_RESETDONE        0x00000001

#define AM335X_TIMER1MS_TISR_MAT_IT_FLAG        0x00000001
#define AM335X_TIMER1MS_TISR_OVF_IT_FLAG        0x00000002
#define AM335X_TIMER1MS_TISR_TCAR_IT_FLAG       0x00000004

#define AM335X_TIMER1MS_TIER_MAT_IT_ENA         0x00000001
#define AM335X_TIMER1MS_TIER_OVF_IT_ENA         0x00000002
#define AM335X_TIMER1MS_TIER_TCAR_IT_ENA        0x00000004
```

ARM Clock Definitions (Part 3)

```
#define AM335X_TIMER1MS_TCLR_ST                 0x00000001
#define AM335X_TIMER1MS_TCLR_AR                 0x00000002

#define AM335X_TIMER1MS_CLKCTRL_ADDR            0x44E004C4
#define AM335X_TIMER1MS_CLKCTRL_EN              0x00000002
```

Clock Interrupt Handler Code (Part 1)

```
/* clkhandler.c - clkhandler */

#include <xinu.h>

/*------------------------------------------------------------------------
 * clkhandler - high level clock interrupt handler
 *------------------------------------------------------------------------
 */
void    clkhandler()
{


        volatile struct am335x_timer1ms *csrptr =
                        (struct am335x_timer1ms *)0x44E31000;
                        /* Set csrptr to address of timer CSR       */

        /* If there is no interrupt, return */

        if((csrptr->tisr & AM335X_TIMER1MS_TISR_OVF_IT_FLAG) == 0) {
                return;
        }
```

Clock Interrupt Handler Code (Part 2)

```
/* Acknowledge the interrupt */

csrptr->tisr = AM335X_TIMER1MS_TISR_OVF_IT_FLAG;

/* Increment 1000ms counter */

count1000++;

/* After 1 sec, increment clktime */

if(count1000 >= 1000) {
        clktime++;
        count1000 = 0;
}
```

Clock Interrupt Handler Code (Part 3)

```
        /* check if sleep queue is empty */

        if(!isempty(sleepq)) {

                /* sleepq nonempty, decrement the key of */
                /* topmost process on sleepq             */

                if((--queuetab[firstid(sleepq)].qkey) == 0) {

                        wakeup();
                }
        }

        /* Decrement the preemption counter */
        /* Reschedule if necessary          */

        if((--preempt) == 0) {
                preempt = QUANTUM;
                resched();
        }
}
```

Clock Initialization (Part 1)

```
/* clkinit.c - clkinit (BeagleBone Black) */

#include <xinu.h>

uint32  clktime;                /* Seconds since boot                   */
uint32  count1000;              /* ms since last clock tick             */
qid16   sleepq;                 /* Queue of sleeping processes          */
uint32  preempt;                /* Preemption counter                   */

/*------------------------------------------------------------------------
 * clkinit  -  Initialize the clock and sleep queue at startup
 *------------------------------------------------------------------------
 */
void    clkinit(void)
{
        volatile struct am335x_timer1ms *csrptr =
        (volatile struct am335x_timer1ms *)AM335X_TIMER1MS_ADDR;
                        /* Pointer to timer CSR in BBoneBlack   */
        volatile uint32 *clkctrl =
                        (volatile uint32 *)AM335X_TIMER1MS_CLKCTRL_ADDR;

        *clkctrl = AM335X_TIMER1MS_CLKCTRL_EN;
        while((*clkctrl) != 0x2) /* Do nothing */ ;
```

Clock Initialization (Part 2)

```
/* Reset the timer module */

csrptr->tiocp_cfg |= AM335X_TIMER1MS_TIOCP_CFG_SOFTRESET;

/* Wait until the reset is complete */

while((csrptr->tistat & AM335X_TIMER1MS_TISTAT_RESETDONE) == 0)
        /* Do nothing */ ;

/* Set interrupt vector for clock to invoke clkint */

set_evec(AM335X_TIMER1MS_IRQ, (uint32)clkhandler);

sleepq = newqueue();    /* Allocate a queue to hold the delta   */
                        /*   list of sleeping processes         */

preempt = QUANTUM;      /* Set the preemption time              */

clktime = 0;            /* Start counting seconds               */
count1000 = 0;
/* The following values are calculated for a    */
/*   timer that generates 1ms tick rate         */

csrptr->tpir = 1000000;
csrptr->tnir = 0;
csrptr->tldr = 0xFFFFFFFF - 26000;
```

Clock Initialization (Part 3)

```
        /* Set the timer to auto reload */

        csrptr->tclr = AM335X_TIMER1MS_TCLR_AR;

        /* Start the timer */

        csrptr->tclr |= AM335X_TIMER1MS_TCLR_ST;

        /* Enable overflow interrupt which will generate */
        /*   an interrupt every 1 ms                     */

        csrptr->tier = AM335X_TIMER1MS_TIER_OVF_IT_ENA;

        /* Kickstart the timer */

        csrptr->ttgr = 1;

        return;
}
```

Notes About Device Hardware Interfaces

- Hardware is incredibly low level
- The interface to a hardware device is tedious
- Hardware defines
  - Many registers that each have some special meaning
  - Special constants that must be used
- A programmer must deal with
  - Silly details
  - A lack of concepts and principles
  - Multiple commands to perform a simple task

Summary

- Two types of timed events are especially important in an operating system
  - Preemption
  - Process delay (sleep)
- A delta list provides an elegant and efficient data structure to store a set of sleeping processes
- If multiple processes awaken at the same time, rescheduling must be deferred until all have been made ready
- *Recvtime* allows a process to specify a maximum time to wait for a message to arrive

Questions?