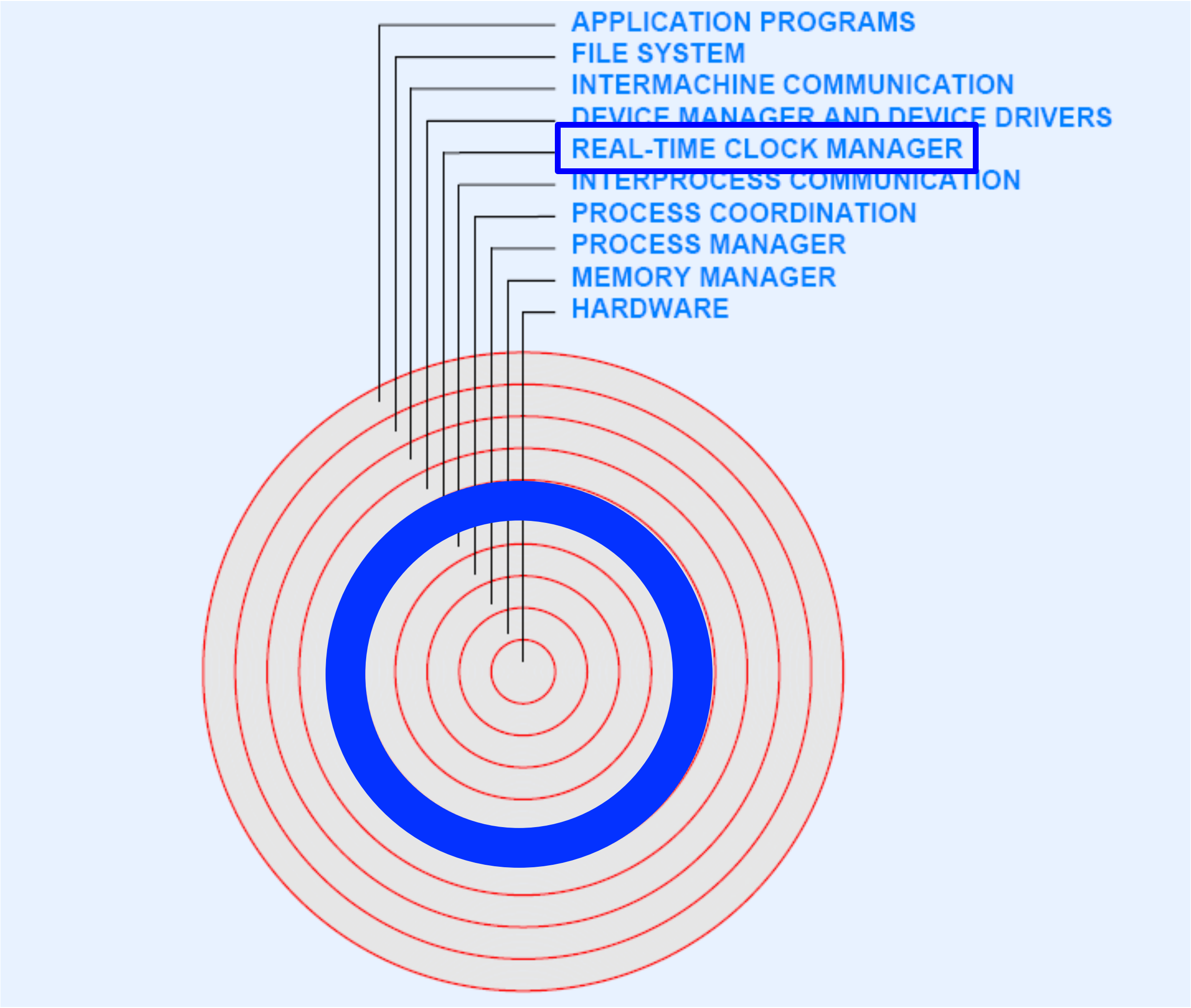
CSCI 8530
Advanced Operating Systems

Part 10

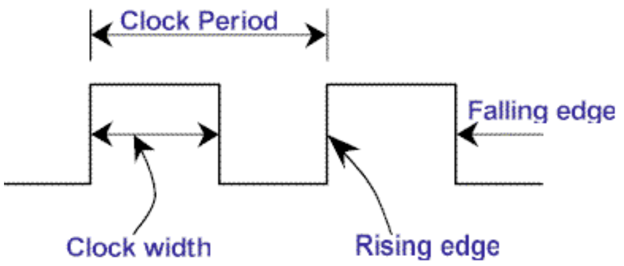
Clock and Timer Management

Location of Device Management in the Hierarchy



Clock Hardware (1/3)

1. Processor clock
   – Controls processor rate
   – Often cited as processor speed
   – Usually not visible to the OS



2. Time-of-day clock (chronometer)
   – Can be set or read by the processor
   – Operates independently from processor

Clock Hardware (2/3)

3. Real-time clock
   - Pulses regularly
   - Called *programmable* if rate can be controlled by OS
   - Interrupts processor on each pulse
   - Does not count pulses; interrupt is lost if missed
     - Means OS cannot leave interrupts disabled for more than one clock tick or real-time clock will be inaccurate (interrupts/pulses will be missed)

   Note: some real-time clock hardware has *counter* for missed interrupts, which allows OS to correct counts later

Clock Hardware (3/3)

4. Interval timer (count-down timer)
   - Hardware device that operates asynchronously
   - An internal real-time clock and a counter
   - Timeout value *T* is set by processor, and device interrupts *T* time units later
   - Processor may be able to find time remaining
   - *Programmable* needs two registers
     - The “period” register: how much time should pass between consecutive interrupts
       - Generate multiple interrupts, coped to “remaining”
     - The “remaining” register: how much time remains before the next interrupt
       - Decremented at a specified rate

TIME MANAGEMENT

Timed Events

- *Hard real-time* system requires each event to occur at (or no later than) an exact time
- *Soft real-time* system requires an event to occur on or *occasionally* after the specified time
- In theory, an OS may need to handle many event types
- In practice, only two basic event types suffice

Two Principle Types of Timed Events

- *Preemption event*
  - Implements *timeslicing* by switching the processor among ready processes
  - Guarantees a given process cannot run forever
  - A preemption event is scheduled during a context switch
  - Cancellation is important (few processes exhaust their timeslice)
- *Sleep event*
  - Move to a new state
  - Scheduled by a process
  - Delays the calling process a specified time

Time-slicing Tradeoff

- How large should a timeslice be?
- Small granularity
  - Advantage: guarantees fairness because processes proceed at approximately equal rate
  - Disadvantage: increased rescheduling overhead
- Large granularity
  - Advantage: lower rescheduling overhead
  - Disadvantage: unfair because one process may be far ahead of another

Timeslicing and Conventional Applications

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.

Without a preemptive capability, an operating system cannot regain control from a process that executes an infinite loop.

Managing Real-time Clock Events

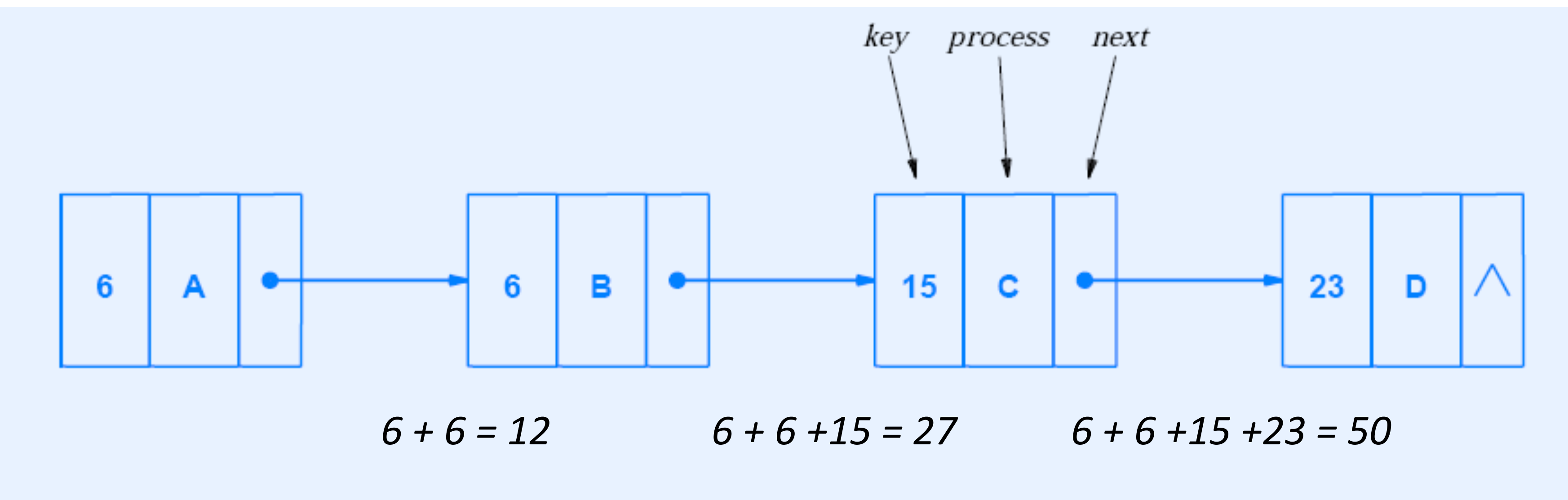
- Must be efficient
  - Clock interrupts occur frequently and continually
  - Set of timed events examined on each clock interrupt
  - Should avoid searching a list (avoid O(N) complexity)
- Mechanism
  - Keep timed events on a linked list
  - One item appears on the list for each outstanding event
  - Called *event queue* (aka *callout queue*)

Delta List

- Data structure used for timed events
- Items on the list are ordered by time of occurrence
- For efficiency, list stores *relative* times
- 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”
- In UNIX, items are called “callouts”

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

- Sleep queue
  - Delta list of delayed processes
  - Each node on the list corresponds to a sleeping process
- Global variable *sleepq* contains the ID of the sleep queue
- Clock interrupt handler
  - Decrements preemption counter
  - Reschedules if timeslice has expired
  - Processes event list

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
- Relationship must be maintained whenever an item is inserted or deleted

Real-time Delay and Clock Resolution

- Process calls *sleep* to delay
- Question: what resolution should be used for sleep?
  - Humans typically think in seconds or minutes
  - Real-time applications may need millisecond accuracy or better (if available)

Xinu Sleep Primitives and Resolution

- Set of primitives accommodates range of possible resolutions

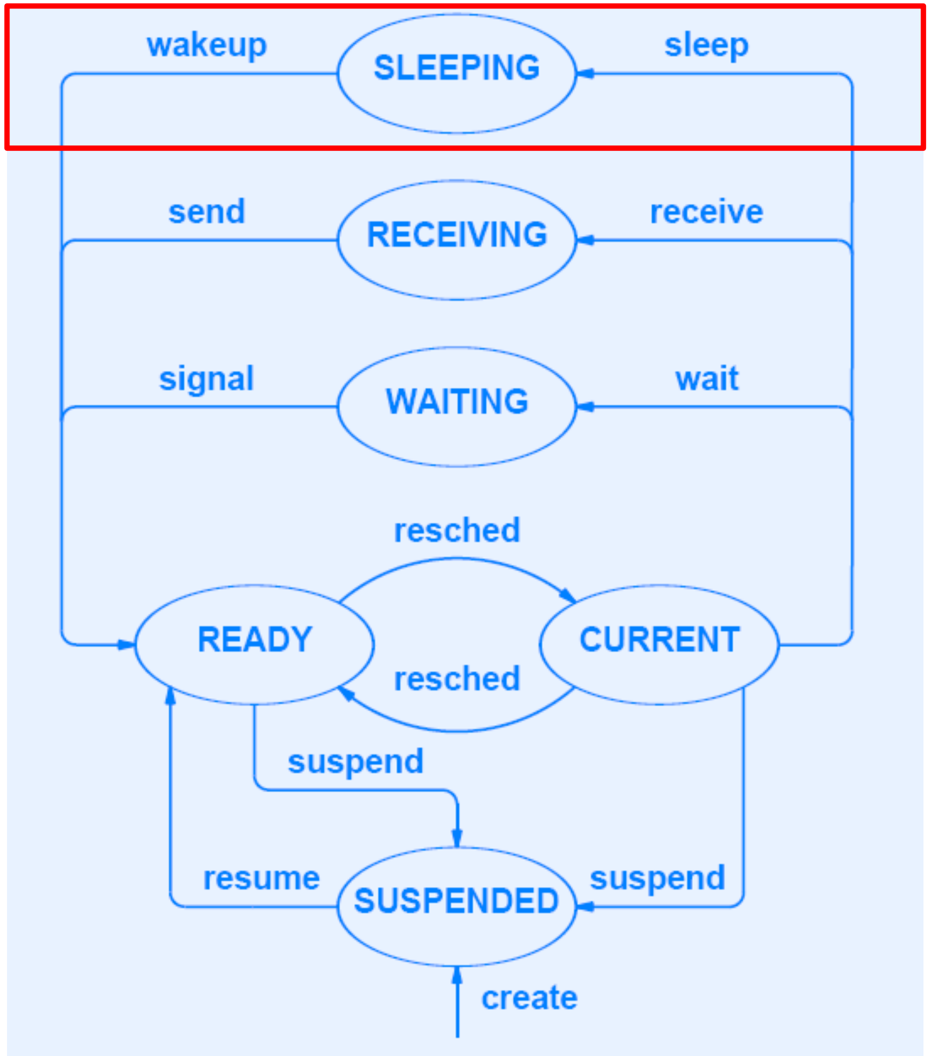
  sleep – delay given in seconds

  sleep10 – delay given in tenths of seconds

  sleep100 – delay given in hundredths of seconds

  sleepms – delay given in milliseconds
- Note: some hardware does not support all resolutions

New Process State for Sleeping Processes



sleepms inserts the process into the delta list of sleeping processes

Xinu Sleep Function (part 1)

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

#include <xinu.h>

#define MAXSECONDS 4294967 /* Max seconds per 32-bit msec */

/*------------------------------------------------------------------------
 * sleep - Delay the calling process n seconds
 *------------------------------------------------------------------------
 */
syscall sleep(
          uint32 delay /* Time to delay in seconds */
        )
{
        if (delay > MAXSECONDS) {
                return SYSERR;
        }
        sleepms(1000*delay);
        return OK;
}
```

Xinu Sleep Function (part 2)

```
/*------------------------------------------------------------------------
 * sleepms - Delay the calling process n milliseconds
 *------------------------------------------------------------------------
 */
syscall sleepms(
          uint32 delay /* Time to delay in msec. */
        )
{
        intmask mask; /* Saved interrupt mask */

        mask = disable();
        if (delay == 0) {
                yield();
                restore(mask);
                return OK;
        }
        /* Delay calling process */
        if (insertd(currpid, sleepq, delay) == SYSERR) {
                restore(mask);
                return SYSERR;
        }
        proctab[currpid].prstate = PR_SLEEP;
        resched();
        restore(mask);
        return OK;
}
```

sleepms uses *insertd* to insert the current process in the delta list of sleeping processes

Given by argument *key*

Inserting an Item on Sleepq

- Current process calls *sleepms* or *sleep* to request delay
- *Sleepms*
  - 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 place to insert the process
  - Adjust remaining keys as necessary

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;
        }
```

Scan the delta list

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;
}
```

Specify a delay relative to the time at which the predecessor of "next" awakens

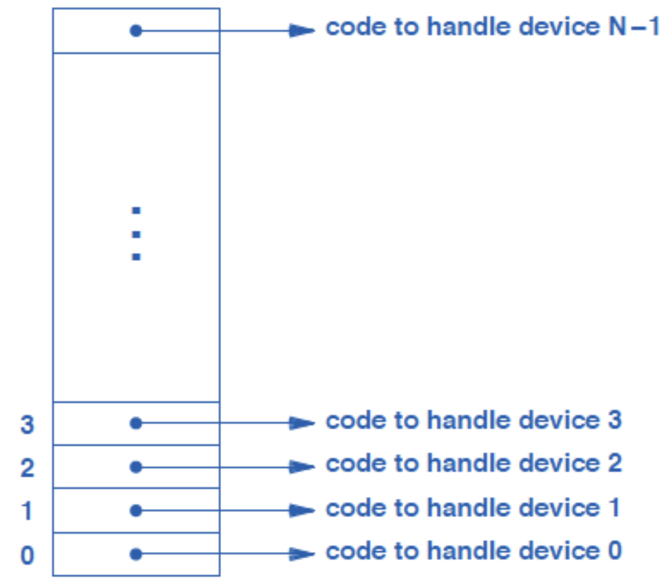
Insertd must also subtract the extra delay that the new item introduces from the delay of the rest of the list

Invariant During Sleepq Insertion

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.

Testing a Clock

- On some systems, clock hardware is optional
- If optional, OS can test for presence of a clock
  - Initialize clock interrupt vector
  - Enable interrupts
  - Loop “long enough”
  - If interrupt occurs, declare clock present
  - Otherwise, declare no clock present and disable *sleep*



Interrupt vectors

Clock Interrupt Handler

- May need to be optimized
- Decrements preemption counter
  - Calls *resched* if counter reaches zero
- Decrements count of the first element on clock queue
  - Calls *wakeup* if counter reaches zero
- Important notes
  - More than one process may awaken at the same time
  - *Wakeup* should awaken 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;
}
```

- Rescheduling is deferred until all processes are awakened

Operation Timeout

- Many operating systems components need a “timeout” feature
- Especially useful in building communication protocols
- Possible approaches
  - Add timeout feature to each system call (difficult)
  - Provide a single facility for timeout
- Xinu uses latter approach with timed message reception

Timed Message Reception

- Implemented with *recvtime()*
- Argument specifies maximum delay
- A call to *recvtime()* returns
  - If a message arrives before the specified timeout
  - After the specified timeout if no message has arrived
- Value *TIMEOUT* is returned if time expires
- Note: timer is cancelled if message arrives

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;          /* 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) { /* If message waiting, no delay */
            if (insertd(currpid,sleepq,maxwait) == SYSERR) {
                    restore(mask);
                    return SYSERR;
            }
            prptr->prstate = PR_RECTIM;  /* A receive with timeout */

            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;
}
```

Example Clock Interrupt Handler (part 1)

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

#include <xinu.h>

/*------------------------------------------------------------------------
 * clkhandler - high level clock interrupt handler
 *------------------------------------------------------------------------
 */
void clkhandler()
{
        static uint32 count1000 = 1000; /* Count to 1000 ms             */

        /* Decrement the ms counter, and see if a second has passed */

        if((--count1000) <= 0) {

                /* One second has passed, so increment seconds count */

                clktime++; // provide the date (e.g., it is used by the
                           //   Xinu shell command date)

                /* Reset the local ms counter for the next second */
                count1000 = 1000;  // counts from 1000 down to 0

        }
```

Manages sleeping processes and Preemption

Store the time in seconds since the system booted

Example Clock Interrupt Handler (part 2)

```
        /* Handle sleeping processes if any exist */

        if(!isempty(sleepq)) {

                /* Decrement the delay for the first process on the     */
                /* sleep queue, and awaken if the count reaches zero    */

                if((--queuetab[firstid(sleepq)].qkey) <= 0) {
                        wakeup();
                }
        }

        /* Decrement the preemption counter, and reschedule when the    */
        /* remaining time reaches zero                                  */

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

Remove all processes from the sleep queue that have a zero delay

Summary (1/2)

- Computer can contain several types of hardware clocks
  - Processor
  - Time of day
  - Real-time
  - Interval timer
- Real-time clock or interval timer used for
  - Preemption
  - Process delay
- OS may need to convert hardware pulse rate to appropriate tick rate

Summary (2/2)

- Delta list provides elegant data structure to store a set of sleeping processes
- Only the key of the first item on the list needs to be updated on each clock tick
- Multiple processes may awaken at the same time; rescheduling is deferred until all have been made ready
- *Recvtime* allows a process to wait a specified time for a message to arrive