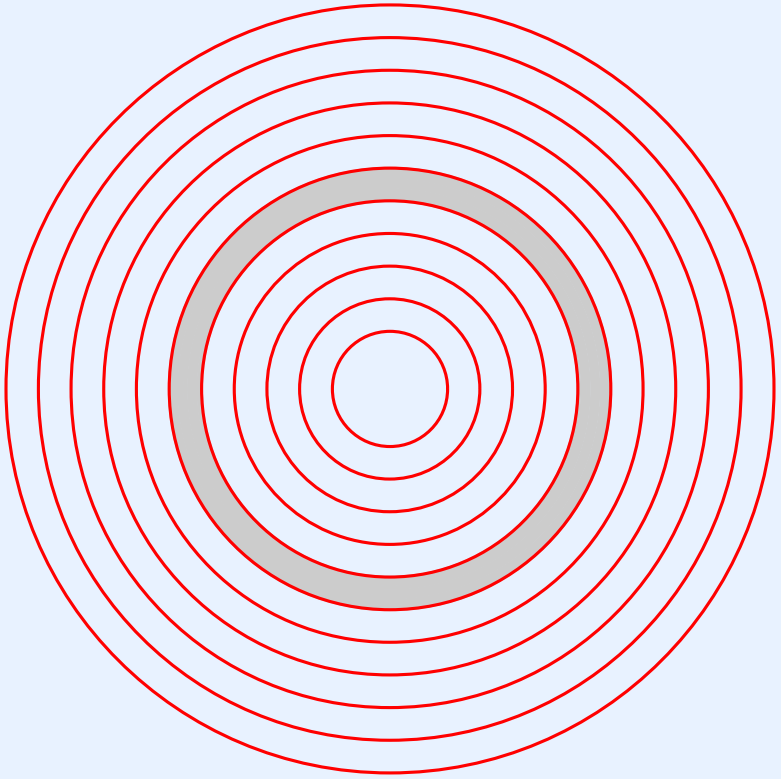
PART 9

Clock And Timer Management

Location Of Clock Management In The Hierarchy



Types Of Clocks

- CPU clock
- Time-of-day clock
- Real-time clock
- Interval timer

CPU Clock

- Controls processor rate
- Often cited as processor speed
- Usually not visible to the OS

Time-Of-Day Clock

- Autonomous chronometer
- Can be set or read by CPU
- Usually operates independently from processing

Real-Time Clock

- Pulses regularly
- Rate may not be controllable by CPU
- Interrupts CPU on each pulse
- Does not count pulses
  - Interrupt lost if missed
- Consequence: CPU cannot run more than one clock tick with interrupts disabled or real-time clock is inaccurate
- Note: some real-time clock hardware has counter that records ticks, so OS can learn about missed ticks

Interval Timer

- Hardware device that operates asynchronously
- Timeout value set by CPU (e.g., time T)
- Device
  - Interrupts T time units later
  - Remains disabled after interrupt until reset
- Advanced features available on *some* hardware
  - CPU can poll to find time remaining
  - Timer value can be reset while timer is running

Use Of Real-Time Clock

- In theory
  - OS may need arbitrary event types
  - Events should occur at exact times (hard real-time constraints)
- In practice
  - Only two basic event types
  - Events occur near specified time (soft real-time constraints)

Two Principle Event Types

- *Preemption event*
  - Implements timeslicing
  - Guarantees process does not run forever
  - Next preemption event scheduled during context switch
  - Cancellation important
- *Sleep event*
  - Scheduled by a process
  - Delays the process a specified time

Time-Slicing Trade-Off

- How large should a timeslice be?
- Small granularity
  - More rescheduling overhead
  - Guarantees fairness because processes proceed at approximately equal rate
- Large granularity
  - Less rescheduling overhead
  - 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.

Hardware Pulses And Clock Ticks

- Real-time clock hardware
  - Has fixed *pulse rate*
  - May run off processor clock
  - Rate may not divide a second by power of ten
- Software
  - Written to use *abstract tick rate*
  - May differ from pulse rate
- Clock interrupt handler matches the two rates

Managing Real-Time Clock Events

- Must be efficient
  - Events examined on each clock interrupt
  - Clock interrupts continuously
  - Should avoid searching a list
- Mechanism
  - Keep events on linked list
  - One item on list for each outstanding event
  - Called *event queue*

Delta List

- List of timed events
- Ordered by time of occurrence
- For efficiency, store *relative* times
- Key in item stores difference (*delta*) between the time for event and time for previous event
- Key in first event stores time from “now”

Delta List Example

- Assume events will occur *17*, *27*, *28*, and *32* ticks from now
- Delta keys are *17*, *10*, *1*, and *4*

Real-Time Clock Processing In Xinu

- Clock interrupt handler
  - Decrements preemption counter
  - Reschedules if timeslice has expired
  - Processes event list
- Clock queue
  - Delta list of delayed processes
  - Each item corresponds to one process
  - Global variable *clockq* specifies head

Keys On The Xinu Clock Queue

- Processes on *clockq* 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 when items are inserted or deleted

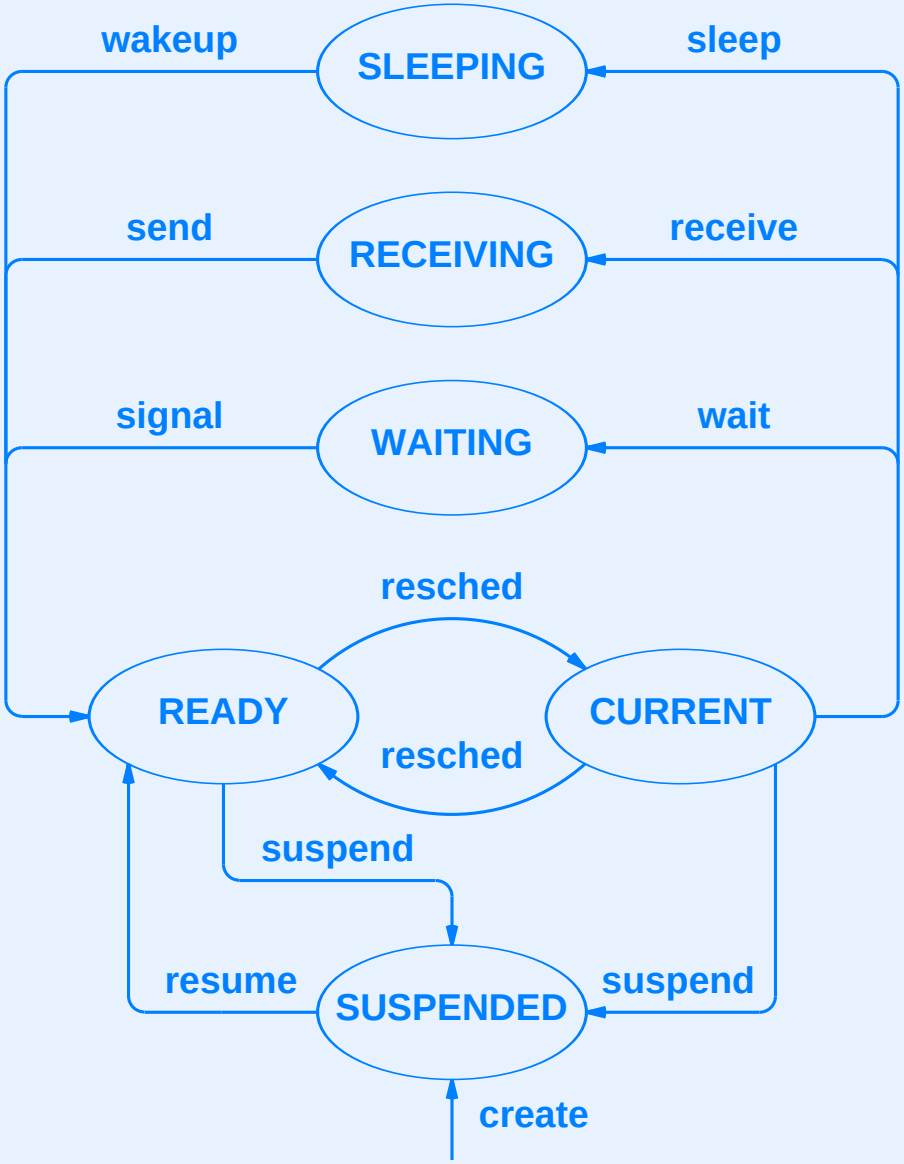
Real-Time Delay And Clock Resolution

- Process calls *sleep* to delay for a specified time
- Original Xinu hardware used 60 Hz real-time clock
  - Sleep times were measured in seconds
  - Smallest resolution was tenths of seconds
- Modern Xinu system uses high-resolution timer hardware
  - Processes can specify delay in milliseconds
- Question: what resolution should be used for sleep?

Xinu Sleep Primitives

- Set of primitives accommodates range of possible resolutions
- Sleep specifies delay in seconds
- Sleep10 specifies delay in tenths of seconds
- Sleep100 specifies delay in hundredths of seconds
- Sleep1000 specifies delay in milliseconds
- Given system may not support all primitives

Process State For Delayed 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)

```
/* 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;
}
```

Inserting An Item On Clockq

- Current process
  - Specifies delay in seconds
- Sleep function
  - Computes delay in milliseconds
  - Inserts current process on *clockq*
  - Calls *resched* to allow other processes to execute
- Method
  - Walk through *clockq* (assumes 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.)    */
        )
{
        int     next;                   /* runs through the delta list  */
        int     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;
}
```

Invariant During Clockq 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.

Clock Initialization

- Clock hardware can be optional
- If optional, OS must 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*

Clock Interrupt Handler

- Highly optimized (assembly code)
- 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* must 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 */

        while (nonempty(sleepq) && (firstkey(sleepq) <= 0)) {
                ready(dequeue(sleepq), RESCHED_NO);
        }
        resched();
        return;
}
```

- Rescheduling is deferred until all processes are awakened

Operation Timeout

- Many operating systems facilities need “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* 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; /* table 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;
                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;
}
```

Summary

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

Summary
(continued)

- List of sleeping processes stored in a delta list
- 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