10.1 Kinds of clocks assoc'd with a computer

Three kinds of clocks are assoc'd with a computer:

1. Central system clock, controlling the rate the CPU executes instructions
2. Real-time clock, pulsing regularly an integral number of times each second, signalling the CPU each time a pulse occurs by posting an interrupt
3. Time-of-day clock, a chronometer like a wristwatch; the CPU controls this clock.

- Unlike the t-o-d clock, a real-time clock does not contains a counter, does not accumulate interrupts (left to the system), and controls the CPU.
- Responsibility for counting interrupts falls upon the system:
  If the CPU takes too long to service a real-time clock interrupt, or if it operates for more than one clock cycle with interrupts disabled, it will miss the next r-t interrupt
  - Systems should service clock interrupts quickly
  - Clock interrupts have highest priority
- Even so, the 11/2 processor cannot call a C-procedure on each clock interrupt, or it would spend most of the time handling them

10.2 Slowing down of the clock rate

How can the clock rate be "effective" adjusted to prevent the system spending most of the time handling interrupts ?

- The clock interrupt (handler/dispatcher) *clkint* simulates a slower-rate clock by dividing the clock rate.
- The LSI 11/2 r-t clock generates 60 pulses per second.
  Clkint "ignores" 5 clock interrupts in a row before (very quickly!) before processing the 6-th one.
- This reduce the "effective" clock rate to 10 pulses per second, the so-called tick rate.

10.3 The use of real-time clock

- OS's use r-t clocks to compute the time-slices allotted for the exec. of each process, by scheduling a preemption event.
- The OS also uses the r-t clock to provide processes with timed delays:

  The System maintains a list of processes, ordered by the time they should be awakened. When the r-t clock in interrupts, it examines this list and wakes up processes for witch the delay has expired.
- A preemption event is used to prevent to processes from running forever. It is set in *resched* by :

  $$\text{preempt} := \text{QUANTUM}$$
- The clock interrupt dispatcher *clkint* decrements preempt on each tick, calling *resched* when preempt = 0.
- QUANTUM shouldn't be too small (e.g. 2 or 3) causing too much overhead, and not too large, slowing down the reaction time.
- In practise most processes are stopped earlier that QUANTUM ticks, by executing *wait* or doing I/O processing.

10.4 Delta List Processing

- To avoid searching through lists, delayed processes are put on **delta list**, residing in the q-structure.
- Variable *clockq* contains q-index of its head
- At each clock tick, *clkint* examines the first process in *clockq*, and calls a high-level interrupt routine *wakeup* to awake processes when their delay has expired.
- Processes on the *clockq* are ordered by the time at witch they should be awakened
  - With each key telling the number of clock ticks witch the process must delay before the preceding one on the *clockq*
- Procedure*insertd(pid, head, key)* inserts process *pid* in delta-list *head*, given its key *key*

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

/*------------------------------------------------
 * insertd -- insert process pid in delta list "head"
 *------------------------------------------------
 */
insertd(pid, head, key)
        int     pid;
        int     head;
        int     key;
{
        int     next;       /* runs through */
        int     prev;       /* follows next */
        for(prev=head, next=q[head].qnext ;
            q[next].qkey < key ; prev=next,next=q[next].qnext)
               key -= q[next].qkey;
        q[pid].qnext = next;
        q[pid].qprev = prev;
        q[pid].qkey  = key;
        q[prev].qnext = pid;
        q[next].qprev = pid;
        if (next < NPROC)
                q[next].qkey -= key;
        return(OK);
}
```

10.5 Putting a process to sleep

- System call *sleep(n)* delay the calling (current) process for n tenth-of-a-second, by moving the current process to the delta list *clockq*
- This requires introducing a new process state for that moved process: **SLEEPING**- see figure 10.1

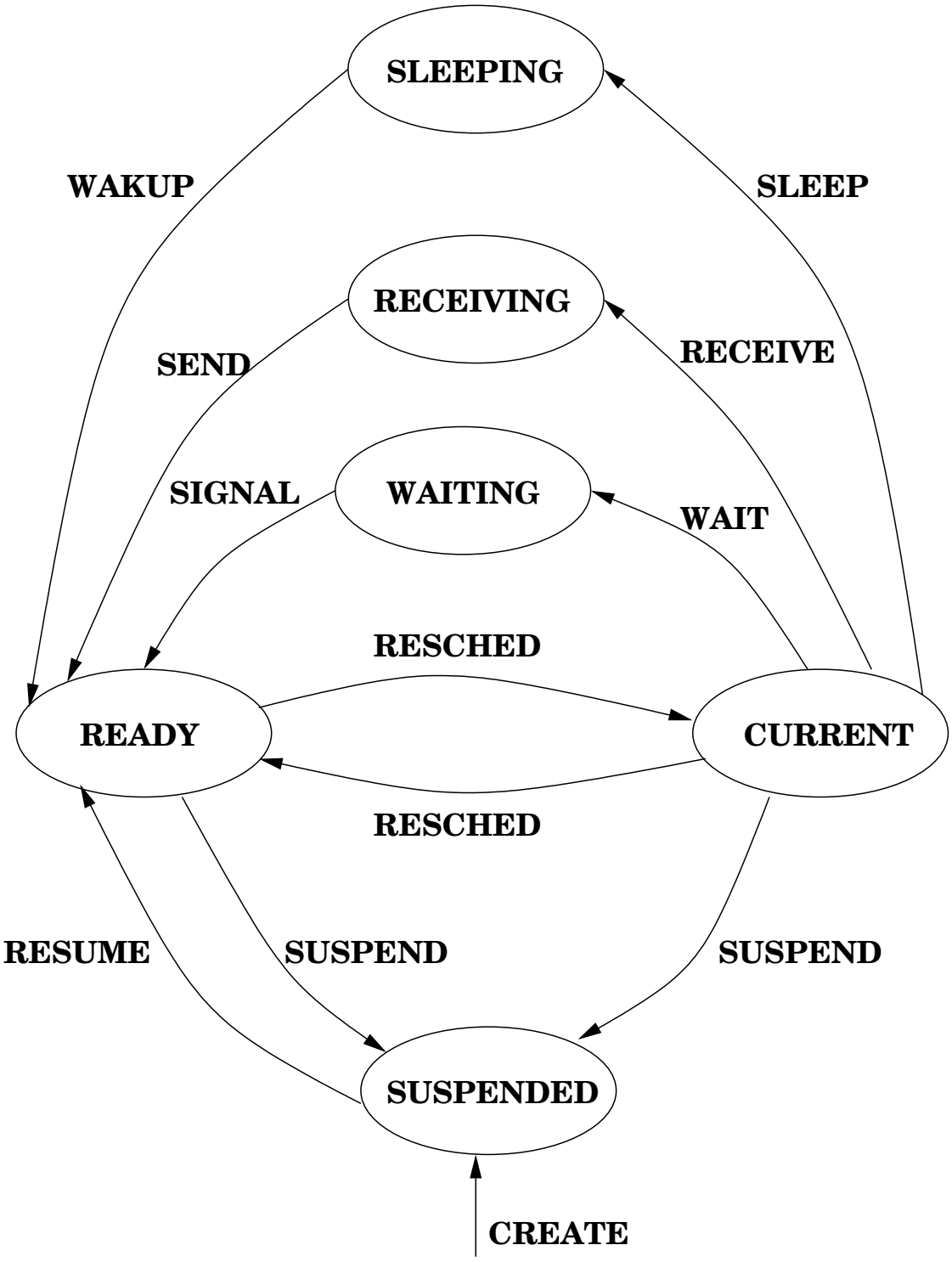

Figure 7.1: The Process state transitions for the sleep state

```
/* sleep10.c - sleep10 */

#include <conf.h>
#include <kernel.h>
#include <proc.h>
#include <q.h>
#include <sleep.h>

/*------------------------------------------------
 * sleep10 -- delay the caller for a time specified
 * in tenths of second
 *------------------------------------------------
 */
SYSCALL sleep10(n)
        int n;
{
        char ps;

        if(n < 0 || clkurns==0)
                return(SYSERR);
        if (n == 0){
                    /* sleep10(0) -> end time slice */
        resched();
                return(OK);
        }
```

```
        disable(ps);
        insertd(currpid,clockq,n);
        slnempty = TRUE;
        sltop = & q[q[clockq].qnext].qkey;
        proctab[currpid].pstate = PRSLEEP;
        resched();
        restore(ps);
        return(OK);
}
```

```
/* sleep.h */

#define CVECTOR 0100     /* location of clock interrupt vector */

extern  int     clkruns; /* 1 iff clock exists; 1 otherwise    */
                         /* Set at system startup.             */
extern  int     clockq;  /* q index of sleeping process list   */
extern  int     count6;  /* used to ignore 5 of 6 interrupts   */
extern  int     *sltop;  /* address of first key on clockq     */
extern  int     slnempty;/* 1 iff clockq is nonempty           */

extern  int     defclk;  /*>0 iff clock interrupts are deferred*/
extern  int     clkdiff; /* number of clock ticks defrred      */
extern  int     clkint(); /* clock interrupt handler           */
```

```
/* setclkr.s - setclkr */

CVECTPC =       100   / clock interrupt vector address
CVECTPS =       102   /  "      "          "       "
DISABLE =       340   / PS that disables interrupts
ENABLE  =       000   / PS taht enables interrupts
COUNT   =       30000 / Times to loop (in decimal)


/*------------------------------------------------------------
 * setclkr -- set clkruns to 1 iff r-t clock exists, 0 otherwise
 *------------------------------------------------------------
```

```
        .globl  _setclkr
_setclkr:
        mov     r1,-(sp)        / save register used
        clr     _clkruns        / initialize for no clock
        mov     *$CVECTPS,-(sp)/ save clock interrupt vector
        mov     *$CVECTPC,-(sp)/  on caller's stack
                                / set up new interrup vector
        mov     $DISABLE,*$CVECTPS
        mov     $setint,*$CVECTPC
        mov     $COUNT,r1       / initialize counter for loop
        reset                   / clear other interrupts, if any
        mtps    $ENABLE         / allow interrupty
setloop:
        dec     r1              / loop COUNT times waiting for
        bpl     setloop         / a clock interrupt
        mtps    $DISABLE        / no interrupt occurred, so quit
        br      setdone
setint:
        inc     _clkruns        / clock interrupt jumps here
        add     $4,sp           / pop pc/ps pushed by interrupt
setdone:
        mov     (sp)+,*$CVECTPC/ restore old interrupt vector
        mov     (sp)+,*$CVECTPS
        mov     (sp)+,r1        / restore register
        rts     PC              T return to caller
```

10.6 Delays Measured in Seconds

- Size of integer n - 16 bits - limits the delay allowed by calling *sleep10(n)* to $2^{15} - 1$ tenths of second = 55 min.
- System call *sleep(n)* measures delays in seconds and so allows delaying up to 9 hours - see the code in *sleep.c*on the next slide.

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

/*------------------------------------------
 * sleep -- delay the calling process n seconds
 *------------------------------------------
 */
SYSCALL sleep(n)
        int     n;
{
        if (n<0 || clkruns==0)
                return(SYSERR);
        if(n === 0){
                resched();
                return(OK);
        }
        while (n >= 1000) {
                sleep10(10*n);
                n -= 1000;
        }
        if (n > 0)
                sleep10(10*n);
        return(OK);
}
```

10.7 Awaking sleeping processes

- Read the code for *clkint* interrupt dispatcher on next slide to see:
  - that *clkint* decrements the count of the first key on *clockq* at each tick, until his equals 0
  - the hight-level interrupt procedure *wakeup()* is called, to put process with key = 0 on the ready list
- *wakeup()* assumes that interrupts have been disabled upon entry.
- read the code for *wakeup()* on the next slide.

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

/*------------------------------------------------
 * wakeup -- called by clock interrup dispatcher
 * to awaken processes
 *------------------------------------------------
 */
INTPROC wakeup()
{
        while (nonempty(clockq) && firstkey(clockq) <= 0)
                ready(getfirst(clockq),RESCHNO);
        if( slnempty = nonempty(clock) )
                sltop = & q[q[clockq].qnext].qkey;
        resched();
}
```

```
/* clkint.s - clkint */

/*------------------------------------------------
 * clkint -- real-time clock interrupt sevice routine
 *------------------------------------------------
 */
       .globl   _clkint
_clkint:
       dec      _count6     / Is this the 6th interrupt ?
       bgt      clret       / no => return
       mov      $6,_count6  / yes=> reset counter&continue
       tst      _defclk     / Are clock ticks deferred ?
       beq      notdef      / no => go process this tick
       inc      _clkdiff    / yes=> count in clkdiff and
       rtt                  / return quickly
notdef:
       tst      _slnempty   / Is sleep queue nonempty?
       beq      clpreem     / no => go process preemption
       dec      *_sltop     / yes=> decrement delta key
       bgt      clpreem     /       on first process,
       mov      r0,-(sp)    /       calling wakeup if
       mov      r1,-(sp)    /       it reaches zero
       jsr      pc,_wakeup  /       (interrupt routine
       mov      (sp)+,r1    /       saves & restore r0
       mov      (sp)+,r0    /       and r1 ;  c doesn't)
```

```
clpreem:
        dec     _preempt    / Decrement preemption counter
        bgt     clret       / and call reasched if it
        mov     r0,-(sp)    / reaches zero
        mov     r1,-(sp)    /    (As before, interrupt
        jsr     pc,_resched /     routine must save &
        mov     (sp)+,r1    /     restore r0 and r1
        mov     (sp)+,r0    /     because c doesn't)
clret:
        rtt                 / Return from interrupt
```

10.8 Deferred clock processing

- The deferred mode allows the system to accumulate clock ticks in variable *ckldiff*
- A process can place the clock in deferred mode by calling *stopclk*, and return clock to real time mode by calling *strtclk*
- *Stopclk* counts deferral requests by incrementing *defclk*
- *strtclk* counts "restarts" requests by decrementing *defclk*
- As long as *defvlk* remains positive the interrupt handler counts clock times in *clkdiff* without processing them
- *strclk* "makes" up for last time when $defclk = 0$, by catching up on all events that should have occurred while the clock remained deferred:
  - *srtclk* updates the preemption counter and
  - abstracts the accumulated ticks from the delay of sleeping process

```
/* ssclock.c - stopclk, strtclk */

#include <conf.h>
#include <kernel.h>
#include <proc.h>
#include <q.h>
#include <sleep.h>

/*------------------------------------------------
 * stopclk -- put the clock in defer mode
 *------------------------------------------------
 */
stopclk()
{
        defclk++;
}
```

```
/*-----------------------------------------------
 * srtclk -- take the clock out of defer mode
 *-----------------------------------------------
 */
srtclk()
{
        char ps;
        int makeup;

        disable(ps);
        if( defclk<=0 || --defclk>0 ){
                restore(ps);
                return;
        }
        makeup = clkdiff;
        preempt -= makeup;
        clkdiff = 0;
        if ( slnempty ){
                for(next=firstid(clockq) ;
                    next < NPROC && q[next].qkey < makeup ;
                    next=q[next].qnext){
                        makeup -=q[next].qkey;
                        q[next].qkey = 0;
                }
```

```
            if ( next < NPROC)
                    q[next].qkey -= makeup;
            wakeup();
        }
        if (preempt <= 0 )
               resched();
        restore(ps);
}
```

10.9 Clock intialization

- The clock interrupt vector should be initialized at system start-up, before clock interrupts occur.
  Done by calling *clkinit()*
- CVECTOR (= 0100) : defined in *sleep.h* on earlier slide
- Recall *setclkr()* initializes *clkruns*
- Read *setclkr()*
  - Observe in setup: hardware clock r1 decreased at tact of hardware clock
  - If after 30.000 ticks of the hardware clock no r-t interrupt occurred, *clkruns* remains zero
  - If clock interrupt occurs, *setclkr*'s control jumps to *setint*: *clkruns* ¿ 0
  - *setclkr* returns, after restoring the original r-t clock vector, to the next instruction pointed at by PC.

```
/* clkinit.c - cklinit */

/*------------------------------------------------
 * clkinit - initialize the clock and sleep queue
 * (called at startup)
 *------------------------------------------------
 */
clkinit()
{
        int  *vector;
        vector = (int *) VECTOR  /* set up interrupt vector */
        *vector++ = clkint;
        setclkr();
        preempt = QUANTUM;    /* initial time quantum       */
        count6 = 6;           /* 60ths of a sec. counter    */
        slnempty = FALSE;   /* initially, no process asleep */
        clkdiff = 0;          /* zero deferred ticks        */
        defclk = 0;           /* clock is not deferred      */
        clockq = newqueue();  /* allocate clock queue in q  */
```