Lab 5

To address changing the scheduler from priority to first come first serve, our approach was to modify the “key” that each new process had when it was inserted into the ready queue. By default, Xinu uses the priority, so when insert() is called on a process it is added to the queue based on its priority relative to the other processes already in the queue.

To achieve a first come first serve scheduler, we modified resched() to call insert() with an “order” argument, instead of priority. We used a global variable (called order) that gets incremented each time a new process is created. Thus, the first process is at the front of the queue, the second after that, and so forth. Then, when a process is called from the ready queue, it is pulled off the queue in order of creation, with no consideration for its priority.

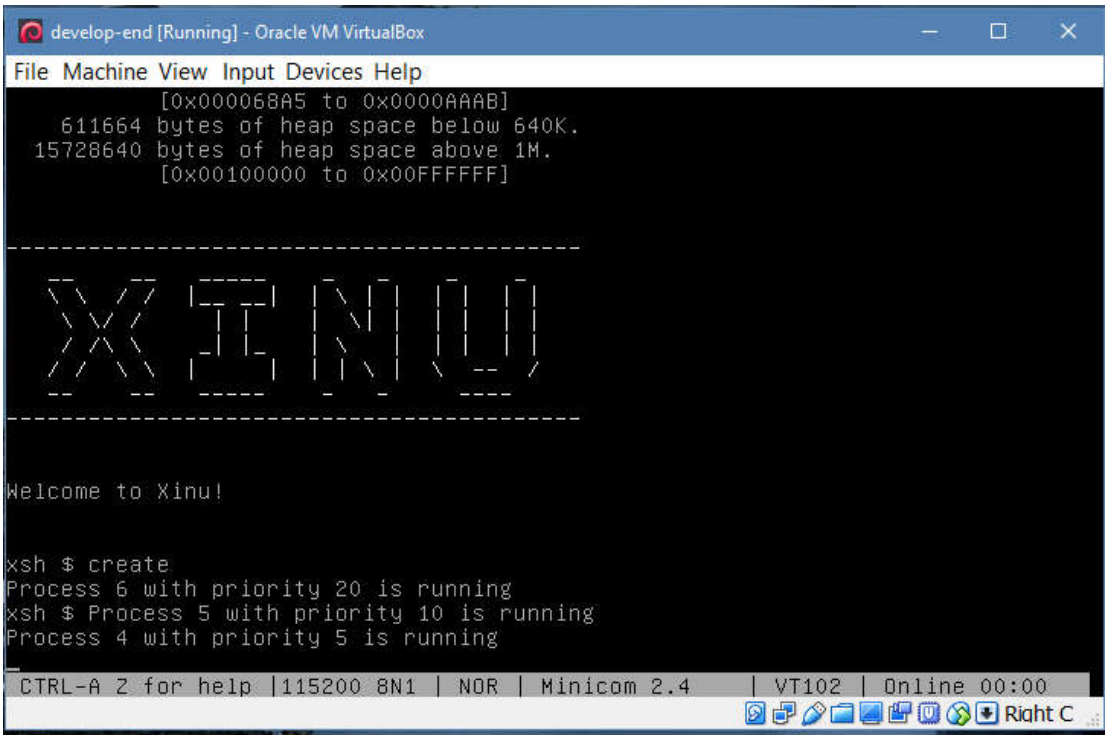
Our actual implementation was to create a new “resume” function (called “resume2”), which was a direct copy of the original resume function except that it calls ready2. The function ready2 is also a direct copy of ready, except that it calls resched2 at the end. Once again, resched2 is an exact copy of resched, except now the global order variable is passed to the insert call instead of priority.

In our shell executable file xsh_create, we added a switch case so both the priority scheduler and first come first serve can be run without any code modification. Simply typing “create” or “create 1” creates three processes and runs them in order of priority. Typing “create 2” creates the same three processes and now runs them in order of creation.

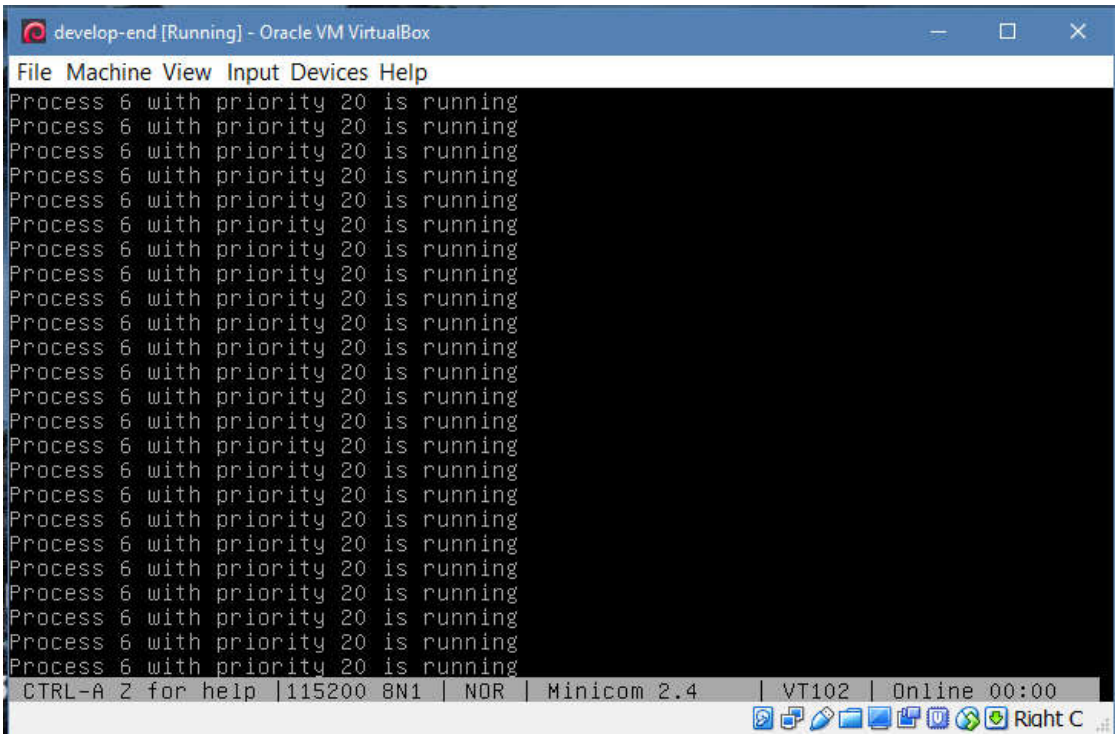
Note that the output is as follows: under condition 1 (priority), the processes are printed in descending priority order (process 6, then 5, then 4). They pause for 15 seconds, and then process 6 runs forever. Under condition 2 (FCFS), the processes are printed in ascending order of creation (process 4, then 5, then 6). They pause for 15 seconds, and then process 4 runs forever.

The processes start at number 4 because PID 0 is the null process, 1 is main, 2 is the shell itself, 3 is the create function from xsh_create shell command, then 4-6 are the newly created processes from our runforever function.

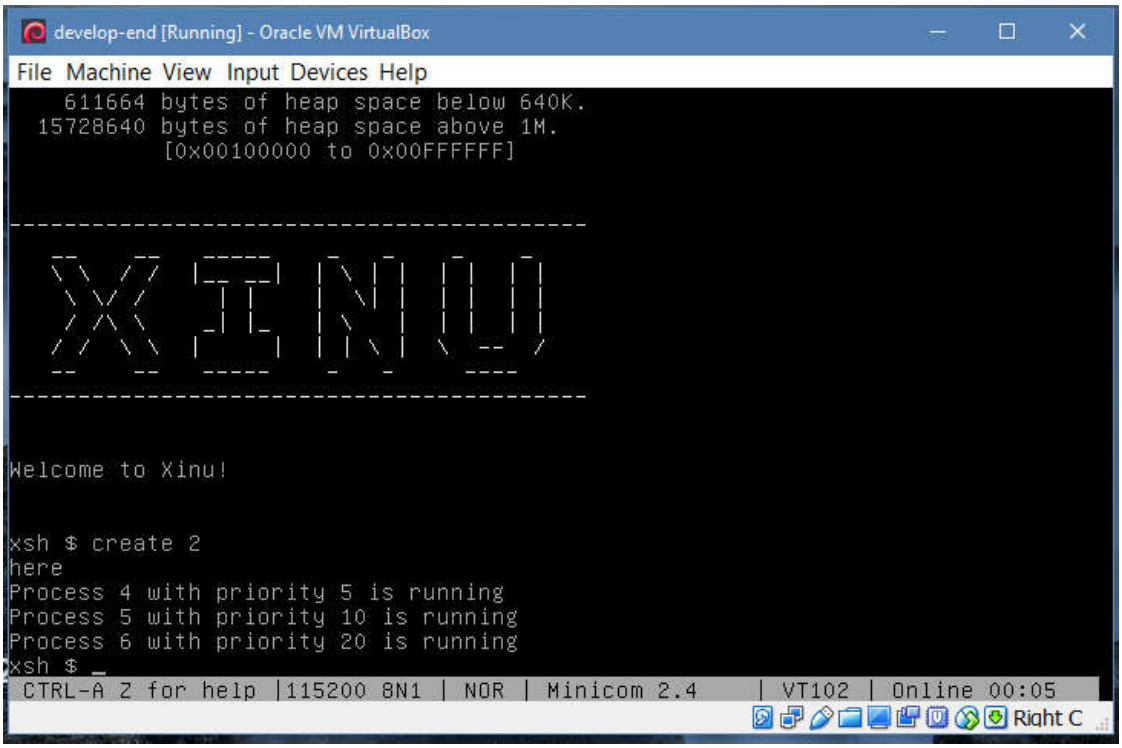
The results of “create 1”: priority-based output:



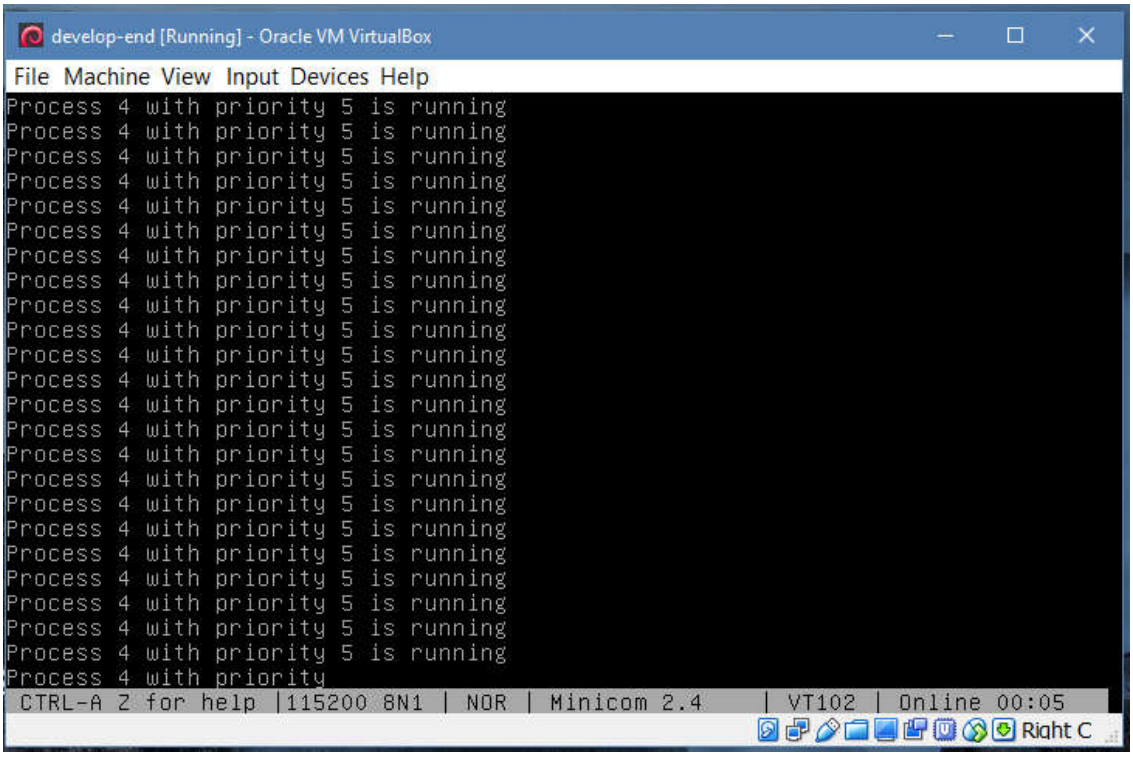
After a 15 second pause:



The results of “create 2”, FCFS:



After 15 second:



Files modified:

Xsh_create

xsh_create.c | runforever.c | main.c | ready2.c | resume2.c | prototypes.h | Makefile | resched2.c

```c
/* xsh_create.c - xsh_create */

#include <xinu.h>
#include <stdio.h>
#include <string.h>

/*------------------------------------------------------------------------
 * xsh_create - shell command to create
 *------------------------------------------------------------------------
 */
shellcmd xsh_create(int nargs, char *args[])
{
    int32   schChoice;
    char    ch;
    char    *chprio;
    pid32   pid1, pid2, pid3;

    if (nargs == 1) {
        schChoice=1;

    }

    else if ( nargs >= 2 ) {
        chprio = args[1];
        ch = *chprio++;
        schChoice = 0;
        while(ch != NULLCH) {
            if ((ch <'0') || (ch > '9')) {
                kprintf("%s: non-digit in entry\n", args[1]);
                return 1;
            }
            schChoice = 10*schChoice + (ch - '0');
            ch = *chprio++;
        }

        if (schChoice < (pri16)MINKEY) {
            kprintf("%s: invalid entry\n", args[1]);
            return 1;
        }

    }
    else {
        kprintf("Too many arguments\n");
        return 1;
    }
    pid1 = create(runforever, 1024, 5, "Process1", 0);
    pid2 = create(runforever, 1024, 10, "Process2", 0);
    pid3 = create(runforever, 1024, 20, "Process3", 0);

    extern int32 order;

    switch(schChoice){
        case 1:
            resume(pid1);
            resume(pid2);
            resume(pid3);
            break;
```

```
41      }
42      else {
43          kprintf("Too many arguments\n");
44          return 1;
45      }
46      pid1 = create(runforever, 1024, 5, "Process1", 0);
47      pid2 = create(runforever, 1024, 10, "Process2", 0);
48      pid3 = create(runforever, 1024, 20, "Process3", 0);
49
50      extern int32 order;
51
52      switch(schChoice){
53          case 1:
54              resume(pid1);
55              resume(pid2);
56              resume(pid3);
57              break;
58          case 2:
59              kprintf("here\n");
60              resume2(pid1);
61              order++;
62              resume2(pid2);
63              order++;
64              resume2(pid3);
65              break;
66          default:
67              kprintf("%s: invalid entry\n", args[1]);
68              break;
69      }
70
71      return 0;
72
73
74
75  }
76
```

Runforever

xsh_create.c | runforever.c | main.c | ready2.c | resume2.c | prototypes.h | Makefile | resched2.c

```
1   #include <xinu.h>
2
3   void runforever(){
4       kprintf("Process %d with priority %d is running\n", getpid(), getprio(getpid()));
5       sleep(15);
6       while(1){
7           kprintf("Process %d with priority %d is running\n", getpid(), getprio(getpid()));
8           //sleep(2);
9       }
10  }
11
```

Main

xsh_create.c | runforever.c | main.c | ready2.c | resume2.c | prototypes.h | Makefile | resched2.c

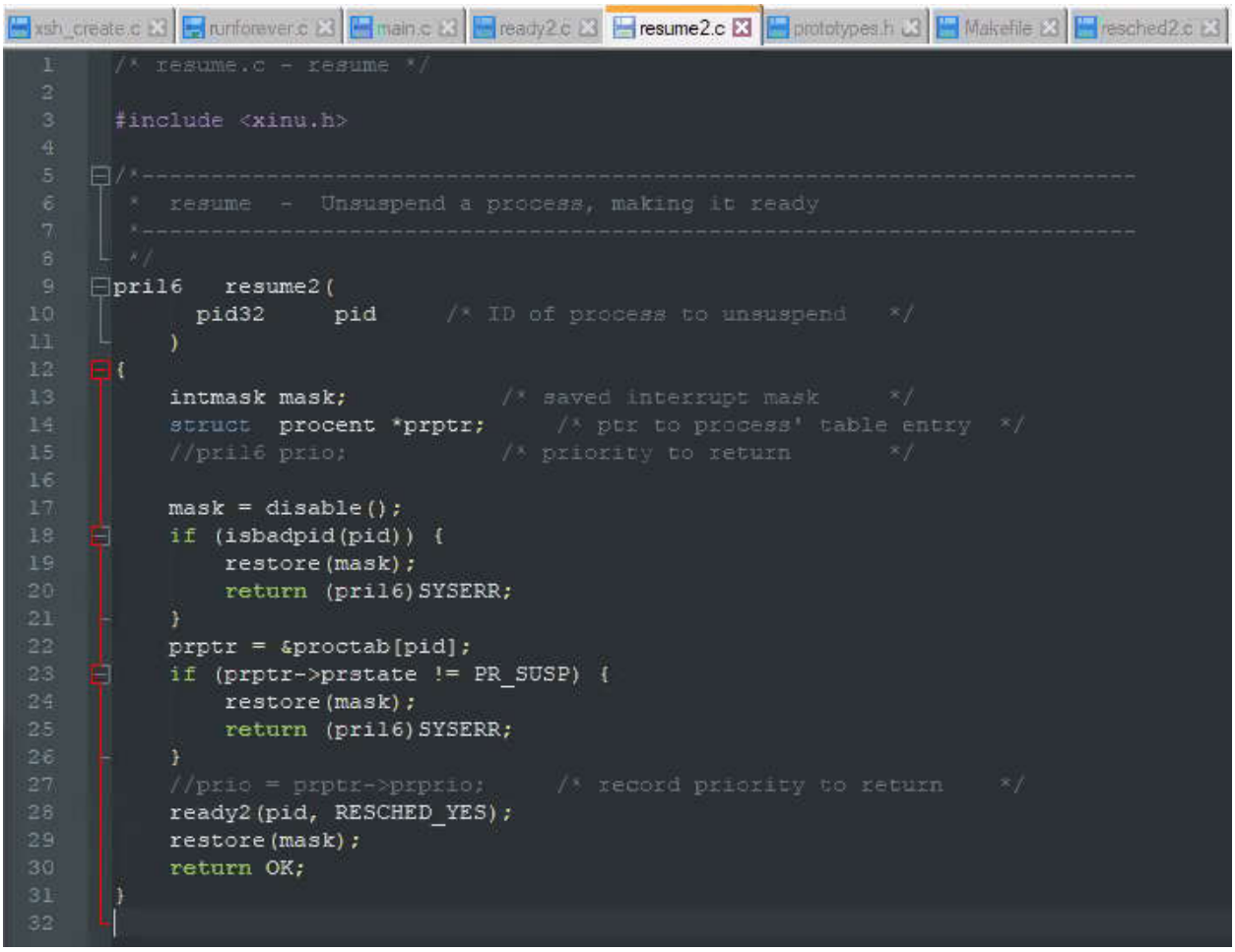
```c
/*  main.c  - main */

#include <xinu.h>
#include <stdio.h>

int32 order;

int main(int argc, char **argv)
{
    uint32 retval;

    order = 1;

    resume(create(shell, 8192, 50, "shell", 1, CONSOLE));

    /* Wait for shell to exit and recreate it */

    recvclr();
    while (TRUE) {
        retval = receive();
        kprintf("\n\n\rMain process recreating shell\n\n\r");
        resume(create(shell, 4096, 1, "shell", 1, CONSOLE));
    }
    while (1);

    return OK;
}
```

Ready2

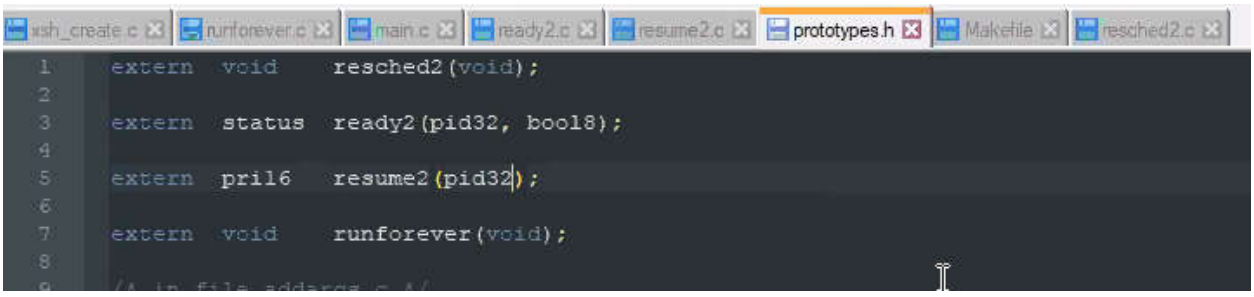
xsh_create.c | runforever.c | main.c | ready2.c | resume2.c | prototypes.h | Makefile | resched2.c

```
/* ready.c - ready */

#include <xinu.h>

qid16   readylist;          /* index of ready list      */

/*------------------------------------------------------------------------
 *  ready  -  Make a process eligible for CPU service
 *------------------------------------------------------------------------
 */
status  ready2(
        pid32       pid,        /* ID of process to make ready  */
        bool8       resch       /* reschedule afterward?    */
      )
{
    register struct procent *prptr;

    if (isbadpid(pid)) {
        return(SYSERR);
    }

    /* Set process state to indicate ready and add to ready list */
    extern int32 order;
    prptr = &proctab[pid];
    prptr->prstate = PR_READY;
    insert(pid, readylist, order);

    if (resch == RESCHED_YES) {
        resched2();
    }
    return OK;
}

```

Resume2



Prototypes



Makefile

xsh_create.c | runforever.c | main.c | ready2.c | resume2.c | prototypes.h | Makefile | resched2.c

```
#------------------------------------------------------------------------

SYSTEM_SFILES =                                 \
    start.S     ctxsw.S     clkint.S    intr.S

SYSTEM_CFILES =                                 \
    ascdate.c   bufinit.c   chprio.c    panic.c     \
    clkinit.c   close.c     conf.c      control.c   \
    create.c    freebuf.c   freemem.c   getbuf.c    \
    getc.c      getdev.c    getitem.c   getmem.c    \
    getpid.c    getprio.c   getstk.c    initialize.c    \
    i386.c      insert.c    insertd.c   ioerr.c     \
    ionull.c    kill.c      kprintf.c   main.c      \
    mkbufpool.c newqueue.c  open.c      pci.c       \
    putc.c      queue.c     read.c      ready.c     \
    receive.c   recvclr.c   recvtime.c  resched.c   \
    resume.c    sched_cntl.c    seek.c      semcount.c  \
    semcreate.c semdelete.c semreset.c  send.c      \
    signal.c    signaln.c   sleep.c     suspend.c   \
    unsleep.c   userret.c   wait.c      wakeup.c    \
    write.c     xdone.c     yield.c     evec.c  runforever.c    \
    ready2.c    resume2.c   resched2.c

SYSTEM_SFULL = ${SYSTEM_SFILES:%=../system/%}
SYSTEM_CFULL = ${SYSTEM_CFILES:%=../system/%}

SRC_FILES += $(SYSTEM_SFULL)
SRC_FILES += $(SYSTEM_CFULL)
```

```
SHELL_CFILES =                                  \
    addargs.c   lexan.c     shell.c

SHELL_CFILES +=                                 \
    xsh_argecho.c   xsh_cat.c   xsh_clear.c     xsh_uptime.c    \
    xsh_echo.c  xsh_exit.c  xsh_devdump.c   xsh_help.c  \
    xsh_kill.c  xsh_memdump.c   xsh_ps.c    xsh_sleep.c     \
    xsh_memstat.c   xsh_create.c

SHELL_CFULL  = ${SHELL_CFILES:%=../shell/%}
```

Resched2


xsh_create.c
runforever.c
main.c
ready2.c
resume2.c
prototypes.h
Makefile
resched2.c
/* resched2.c - resched2 */
#include <xinu.h>
/*--
* resched - Reschedule processor to first come first serve
*--
*/
void resched2(void) /* assumes interrupts are disabled */
{
struct procent *ptold; /* ptr to table entry for old process */
struct procent *ptnew; /* ptr to table entry for new process */
/* If rescheduling is deferred, record attempt and return */
if (Defer.ndefers > 0) {
Defer.attempt = TRUE;
return;
}
/* Point to process table entry for the current (old) process */
ptold = &proctab[currpid];
//if (ptold->prstate == PR_CURR) { /* process remains running */
// if (order > firstkey(readylist)) {
// return;
// }
/* Old process will no longer remain current */
ptold->prstate = PR_READY;
extern int32 order;
insert(currpid, readylist, order);
//}
/* Force context switch to highest priority ready process */
currpid = dequeue(readylist);
ptnew = &proctab[currpid];
ptnew->prstate = PR_CURR;
preempt = QUANTUM; /* reset time slice for process */
ctxsw(&ptold->prstkptr, &ptnew->prstkptr);
/* Old process returns here when resumed */
return;
}