

## 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.

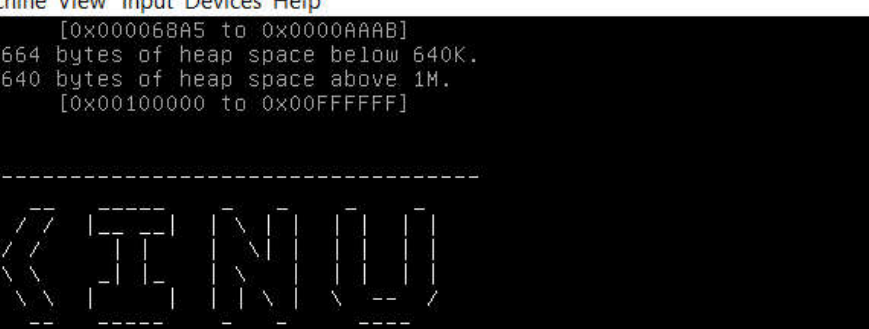
Our actual implementation was to create a new “resume” function (called “`resume2`”), which was a direct copy of the original `resume` function except that it calls `ready2`. The function `ready2` is also a direct copy of `ready`, except that it calls `resched2` at the end. Once again, `resched2` is 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:



The screenshot shows a Windows desktop with a blue taskbar at the top. The active window is titled "develop-end [Running] - Oracle VM VirtualBox". Inside the window is a black terminal window titled "File Machine View Input Devices Help". The terminal output shows the Xinu boot process:

```
[0x00006BA5 to 0x0000AAAB]
611664 bytes of heap space below 640K.
15728640 bytes of heap space above 1M.
[0x00100000 to 0x00FFFFFF]

-----

XINU

-----

Welcome to Xinu!

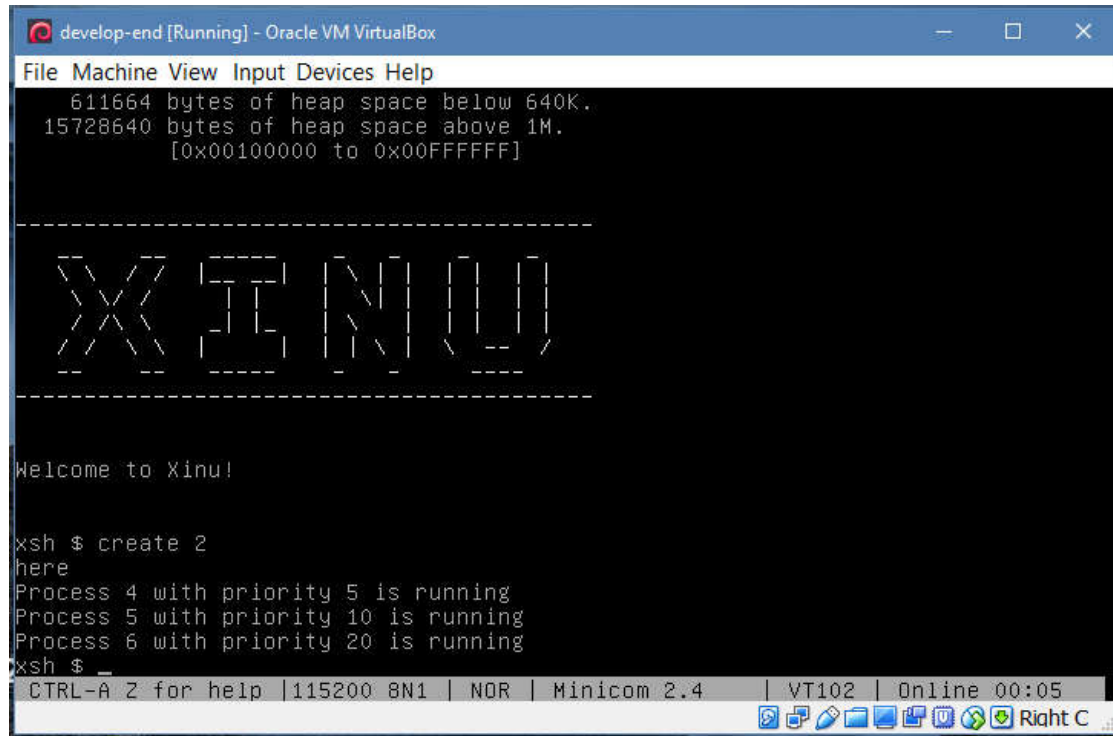
xsh $ create
Process 6 with priority 20 is running
xsh $ Process 5 with priority 10 is running
Process 4 with priority 5 is running
```

The bottom of the terminal window has a status bar with the text "CTRL-A 2 for help | 115200.8N1 | NOR | Minicom 2.4 | VT102 | Online 00:00". On the far right of the status bar, there is a row of icons for various system functions, followed by the text "Right C".

After a 15 second pause:

[illegible]

The results of “create 2”, FCFS:



```
develop-end [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
  611664 bytes of heap space below 640K.
 15728640 bytes of heap space above 1M.
  [0x00100000 to 0x00FFFFFF]

-----
XINU
-----

Welcome to Xinu!

xsh $ create 2
here
Process 4 with priority 5 is running
Process 5 with priority 10 is running
Process 6 with priority 20 is running
xsh $ _
CTRL-A Z for help | 115200 8N1 | NOR | Minicom 2.4 | VT102 | Online 00:05
```

After 15 second:



## Xsh\_create

```

1  /* xsh_create.c - xsh_create */
2
3  #include <xinu.h>
4  #include <stdio.h>
5  #include <string.h>
6
7  /*-----
8   * xsh_create - shell command to create
9   *-----
10  */
11  shellcmd xsh_create(int nargs, char *args[])
12  {
13      int32  schChoice;
14      char   ch;
15      char   *chprio;
16      pid32  pid1, pid2, pid3;
17
18      if (nargs == 1) {
19          schChoice=1;
20      }
21
22      else if ( nargs >= 2 ) {
23          chprio = args[1];
24          ch = *chprio++;
25          schChoice = 0;
26          while(ch != NULLCH) {
27              if ((ch < '0') || (ch > '9')) {
28                  kprintf("%s: non-digit in entry\n", args[1]);
29                  return 1;
30              }
31              schChoice = 10*schChoice + (ch - '0');
32              ch = *chprio++;
33          }
34
35          if (schChoice < (pril6)MINKEY) {
36              kprintf("%s: invalid entry\n", args[1]);
37              return 1;
38          }
39      }
40
41      else {
42          kprintf("Too many arguments\n");
43          return 1;
44      }
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      }
59  }

```

```

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

```

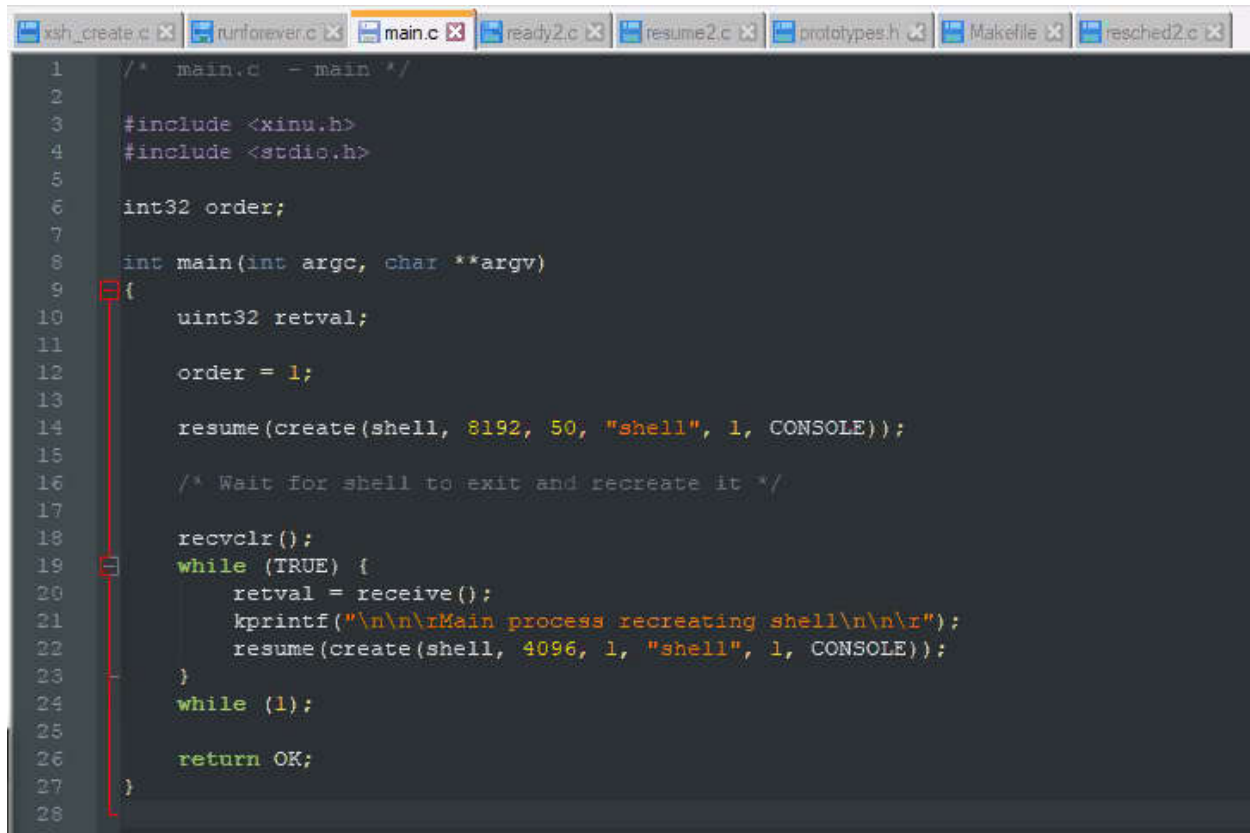
## Runforever

```

xsh_create.c x 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



```
1  /* main.c - main */
2
3  #include <xinu.h>
4  #include <stdio.h>
5
6  int32 order;
7
8  int main(int argc, char **argv)
9  {
10     uint32 retval;
11
12     order = 1;
13
14     resume(create(shell, 8192, 50, "shell", 1, CONSOLE));
15
16     /* Wait for shell to exit and recreate it */
17
18     recvclr();
19     while (TRUE) {
20         retval = receive();
21         kprintf("\n\nrMain process recreating shell\n\nr");
22         resume(create(shell, 4096, 1, "shell", 1, CONSOLE));
23     }
24     while (1);
25
26     return OK;
27 }
28
```



## Ready2

```

1  /* ready.c - ready */
2
3  #include <xinu.h>
4
5  qid16  readylist;      /* index of ready list */
6
7  /*-----
8   * ready - Make a process eligible for CPU service
9   *-----
10 */
11 status ready2(
12     pid32  pid,          /* ID of process to make ready */
13     bool8   resch        /* reschedule afterward? */
14 )
15 {
16     register struct procent *prptr;
17
18     if (isbadpid(pid)) {
19         return(SYSERR);
20     }
21
22     /* Set process state to indicate ready and add to ready list */
23     extern int32 order;
24     prptr = &proctab[pid];
25     prptr->prstate = PR_READY;
26     insert(pid, readylist, order);
27
28     if (resch == RESCHED_YES) {
29         resched2();
30     }
31     return OK;
32 }
33

```



## Resume2

```

1  /* resume.c - resume */
2
3  #include <xinu.h>
4
5  /*-----
6   * resume - Unsuspend a process, making it ready
7   *-----
8   */
9  pril6 resume2(
10     pid32 pid /* ID of process to unsuspend */
11 )
12 {
13     intmask mask; /* saved interrupt mask */
14     struct procent *prptr; /* ptr to process' table entry */
15     //pril6 prio; /* priority to return */
16
17     mask = disable();
18     if (isbadpid(pid)) {
19         restore(mask);
20         return (pril6) SYSERR;
21     }
22     prptr = &proctab[pid];
23     if (prptr->prstate != PR_SUSP) {
24         restore(mask);
25         return (pril6) SYSERR;
26     }
27     //prio = prptr->prprio; /* record priority to return */
28     ready2(pid, RESCHED_YES);
29     restore(mask);
30     return OK;
31 }
32

```

## Prototypes

```

1  extern void resched2(void);
2
3  extern status ready2(pid32, bool8);
4
5  extern pril6 resume2(pid32);
6
7  extern void runforever(void);
8
9  /* in file address.c */

```

## Makefile

```

49 #-----
50
51 SYSTEM_SFILES = \
52     start.S      ctxsw.S      clkint.S      intr.S
53
54 SYSTEM_CFILES = \
55     asctime.c    bufinit.c    chprio.c    panic.c \
56     clkinit.c    close.c      conf.c    control.c \
57     create.c     freebuf.c    freemem.c getbuf.c \
58     getc.c       getdev.c     getitem.c getmem.c \
59     getpid.c     getprio.c    getstk.c  initialize.c \
60     i386.c       insert.c     insertd.c ioerr.c \
61     ionull.c     kill.c       kprintf.c main.c \
62     mbufpool.c   newqueue.c   open.c    pci.c \
63     putc.c       queue.c      read.c    ready.c \
64     receive.c    recvclr.c    recvtime.c resched.c \
65     resume.c     sched_ctl.c   seek.c    semcount.c \
66     semcreate.c  semdelete.c semreset.c send.c \
67     signal.c     signaln.c   sleep.c   suspend.c \
68     unsleep.c    userret.c   wait.c    wakeup.c \
69     write.c      xdone.c     yield.c   evect.c  runforever.c \
70     ready2.c     resume2.c    resched2.c
71
72 SYSTEM_SFULL = ${SYSTEM_SFILES:%=../system/%}
73 SYSTEM_CFULL = ${SYSTEM_CFILES:%=../system/%}
74
75 SRC_FILES += $(SYSTEM_SFULL)
76 SRC_FILES += $(SYSTEM_CFULL)
77

```

```

101 SHELL_CFILES = \
102     addargs.c    lexan.c      shell.c
103
104 SHELL_CFILES += \
105     xsh_argecho.c xsh_cat.c   xsh_clear.c    xsh_uptime.c \
106     xsh_echo.c   xsh_exit.c  xsh_devdump.c  xsh_help.c \
107     xsh_kill.c   xsh_memdump.c xsh_ps.c       xsh_sleep.c \
108     xsh_memstat.c xsh_create.c
109
110 SHELL_CFULL = ${SHELL_CFILES:%=../shell/%}
111

```

## Resched2

```

1  /* resched2.c - resched2 */
2
3  #include <xinu.h>
4
5  /*-----
6   * resched - Reschedule processor to first come first serve
7   *-----
8   */
9  void resched2(void) /* assumes interrupts are disabled */
10 {
11     struct proctent *ptold; /* ptr to table entry for old process */
12     struct proctent *ptnew; /* ptr to table entry for new process */
13
14     /* If rescheduling is deferred, record attempt and return */
15
16     if (Defer.ndefers > 0) {
17         Defer.attempt = TRUE;
18         return;
19     }
20
21     /* Point to process table entry for the current (old) process */
22
23     ptold = &proctab[currpid];
24
25     //if (ptold->prstate == PR_CURR) { /* process remains running */
26     // if (order > firstkey(readylist)) {
27     //     return;
28     // }
29
30     /* Old process will no longer remain current */
31
32     ptold->prstate = PR_READY;
33     extern int32 order;
34     insert(currpid, readylist, order);
35     //}
36
37     /* Force context switch to highest priority ready process */
38
39     currpid = dequeue(readylist);
40     ptnew = &proctab[currpid];
41     ptnew->prstate = PR_CURR;
42     preempt = QUANTUM; /* reset time slice for process */
43     ctxsw(&ptold->prstkptr, &ptnew->prstkptr);
44
45     /* Old process returns here when resumed */
46
47     return;
48 }
49

```