Lab 6

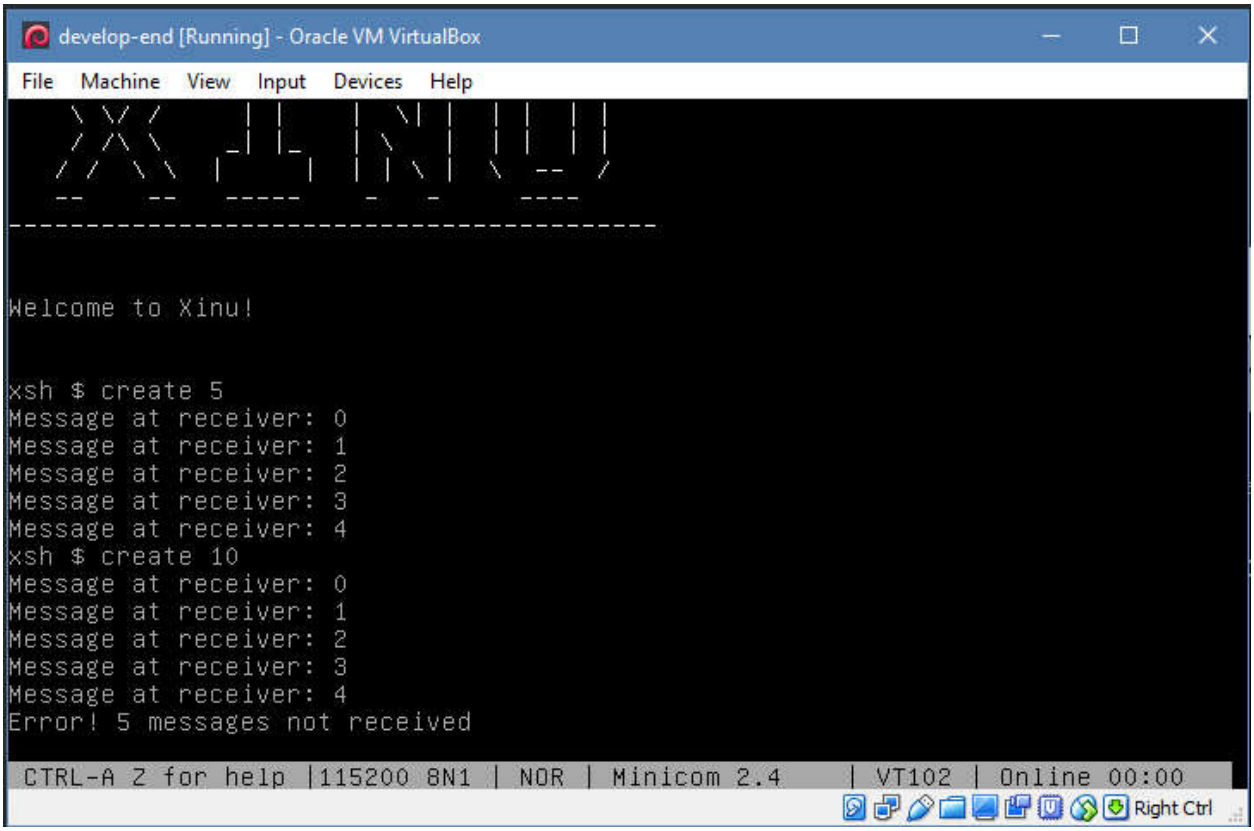
To create a system that can record up to K messages per process, we implemented the port method as described in the textbook. Basically, there is a port data structure setup in the file ports.h that allows for a message buffer, but also checks to make sure there are free ports available. We didn't use the send or receive function, instead opting to use the built-in functions that deal with the port structure (specifically ptinit, ptsend, ptrecv, and ptcreate). These functions allow for the creation and initialization of a given number of ports, as well as access to the global message buffer which used to send the messages.

The process flow was to take an argument from the user to define the number of messages to send. We default the number of ports to 5 (as defined in our single shell function xsh_create). Then, to get the messaging system set up, we first initialize all the ports with ptinit. This sets **all** the ports in Xinu (as defined by the global constant NPORTS which we did not edit) to be in the state PT_FREE. Then, we created the same number of ports using ptcreate. Now, for all the ports to be set to PT_ALLOC, which allows for them to actually receive a message.

At this point, we needed to create a process to receive the message. At its core, all the process does is call ptrecv on the port we send the messages to (we just the first port, port 0). Next we added a loop to call ptrecv for as many messages as we sent. Finally, we built in checks to ensure that the number of messages would not be greater than the number of ports available.

To send the messages, we used the system call ptsend. Each call of ptsend sends one message, so we set a loop for the total messages as defined previously by the user. The loop sends the messages, and the receiver function gathers them and prints them out. In our case, we just the counter integers as our "messages".

Below the screenshot shows first the user entering "create 5" which begins the process and sends 5 messages. Because the number of ports is equal to 5, all 5 messages are sent without error. Next the user enters "create 10" which causes 10 messages to be sent. This time, only 5 are received, and an error calculating the number of undelivered messages is displayed:


develop-end [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Welcome to Xinu!
xsh $ create 5
Message at receiver: 0
Message at receiver: 1
Message at receiver: 2
Message at receiver: 3
Message at receiver: 4
xsh $ create 10
Message at receiver: 0
Message at receiver: 1
Message at receiver: 2
Message at receiver: 3
Message at receiver: 4
Error! 5 messages not received
CTRL-A Z for help |115200 8N1 | NOR | Minicom 2.4 | VT102 | Online 00:00
Right Ctrl

For the relevant screenshots, first the port related functions. These are directly from Xinu, but were not present in our Xinu folders, so we manually added:

ptinit.c

C:\Users\kmartin\Desktop\Pi\Classes\CPS675 OS\vbox\HW6\system\ptinit.c - Notepad++

File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?

prototypes.h | ptinit.c | ptsend.c | ptrecv.c | ptcreate.c | Makefile | process.h | xsh_create.c | main.c | ports.h

```c
/* ptinit.c*/

#include <xinu.h>

struct  ptnode *ptfree;
struct  ptentry porttab[NPORTS];
int32   ptnextid;

/*------------------------------------------------------------------------
 *  intialize all ports
 *------------------------------------------------------------------------
 */
syscall ptinit(
        int32     maxmsgs
      )
{
    int32   i;
    struct  ptnode *next, *curr;

    ptfree = (struct ptnode *)getmem(maxmsgs*sizeof(struct ptnode));
    if (ptfree == (struct ptnode *)SYSERR){
        panic("ptinit failed, insufficient memory");
    }
    for (i=0; i<NPORTS; i++){
        porttab[i].ptstate = PT_FREE;
        porttab[i].ptseq = 0;
    }
    ptnextid = 0;

    for (curr=next=ptfree; --maxmsgs>0; curr=next){
        curr->ptnext = ++next;
    }

    curr->ptnext = NULL;
    return OK;

}
```

ptsend.c

C:\Users\kmartin\Desktop\Pi\Classes\CPS675 OS\vbox\HW6\system\ptinit.c - Notepad++

File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?

prototypes.h | ptinit.c | ptsend.c | ptrecv.c | ptcreate.c | Makefile | process.h | xsh_create.c | main.c | port

```c
/* ptinit.c*/

#include <xinu.h>

struct  ptnode *ptfree;
struct  ptentry porttab[NPORTS];
int32   ptnextid;

/*------------------------------------------------------------------------
 *  intialize all ports
 *------------------------------------------------------------------------
 */
syscall ptinit(
        int32     maxmsgs
    )
{
    int32   i;
    struct  ptnode *next, *curr;

    ptfree = (struct ptnode *)getmem(maxmsgs*sizeof(struct ptnode));
    if (ptfree == (struct ptnode *)SYSERR){
        panic("ptinit failed, insufficient memory");
    }
    for (i=0; i<NPORTS; i++){
        porttab[i].ptstate = PT_FREE;
        porttab[i].ptseq = 0;
    }
    ptnextid = 0;

    for (curr=next=ptfree; --maxmsgs>0; curr=next){
        curr->ptnext = ++next;
    }

    curr->ptnext = NULL;
    return OK;

}
```

ptsend.c

C:\Users\kmartin\Desktop\Pi\Classes\CPS675 OS\vbox\HW6\system\ptsend.c - Notepad++

File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?

prototypes.h | ptinit.c | ptsend.c | ptrecv.c | ptcreate.c | Makefile | process.h | xsh_create.c | main.c | ports.h

```c
/* ptsend.c*/

#include <xinu.h>

/*------------------------------------------------------------------------
 *  send message to port
 *------------------------------------------------------------------------
 */
syscall ptsend(
        int32     portid,
        umsg32    msg
      )
{
    intmask mask;
    int32   seq;
    struct  ptentry *ptptr;
    struct  ptnode  *msgnode;
    struct  ptnode  *tailnode;

    mask = disable();
    if (isbadport(portid) ||
        (ptptr=&porttab[portid])->ptstate != PT_ALLOC){
            restore(mask);
            return SYSERR;
    }

    seq = ptptr->ptseq;
    if (wait(ptptr->ptssem) == SYSERR
    || ptptr->ptstate != PT_ALLOC
    || ptptr->ptseq != seq){
        restore(mask);
        return SYSERR;
    }
    if (ptfree == NULL){
        panic("Port system ran out of message nodes");
    }

    msgnode = ptfree;
    ptfree = msgnode->ptnext;
    msgnode->ptnext = NULL;
    msgnode->ptmsg = msg;

    tailnode = ptptr->pttail;
    if (tailnode == NULL){
        ptptr->pttail = ptptr->pthead = msgnode;
    }else{
        tailnode->ptnext = msgnode;
        ptptr->pttail = msgnode;
    }
    signal(ptptr->ptrsem);
    restore(mask);
    return OK;

}
```

ptrecv.c

C:\Users\kmartin\Desktop\Pi\Classes\CPS675 OS\vbox\HW6\system\ptrecv.c - Notepad++

File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?

prototypes.h | ptinit.c | ptsend.c | ptrecv.c | ptcreate.c | Makefile | process.h | xsh_create.c | main.c

```c
/* ptrecv.c*/

#include <xinu.h>

/*------------------------------------------------------------------------
 *  receive a message from a port
 *------------------------------------------------------------------------
 */
uint32  ptrecv(
        int32     portid
    )
{
    intmask mask;           /* saved interrupt mask     */
    int32   seq;
    umsg32  msg;
    struct  ptentry *ptptr;
    struct  ptnode  *msgnode;

    mask = disable();
    if (isbadport(portid) ||
    (ptptr=&porttab[portid])->ptstate != PT_ALLOC){
        restore(mask);
        return (uint32)SYSERR;
    }

    seq = ptptr->ptseq;
    if (wait(ptptr->ptrsem) == SYSERR || ptptr->ptstate != PT_ALLOC
        || ptptr->ptseq != seq){
            restore(mask);
            return (uint32)SYSERR;
        }

    msgnode = ptptr->pthead;
    msg = msgnode->ptmsg;
    if (ptptr->pthead == ptptr->pttail)
        ptptr->pthead = ptptr->pttail = NULL;
    else
        ptptr->pthead = msgnode->ptnext;
    msgnode->ptnext = ptfree;
    ptfree = msgnode;
    signal(ptptr->ptssem);
    restore(mask);
    return msg;
}
```

Makefile

C:\Users\kmartin\Desktop\Pi\Classes\CPS675 OS\vbox\HW6\compile\Makefile - Notepad++

File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?

prototypes.h | ptinit.c | ptsend.c | ptrecv.c | ptcreate.c | Makefile | process.h | xsh_create.c | main.c | ports.h

```
TOPDIR  = ..

# Components (files for each are listed below)

COMPS   =   system                          \
        device/tty                      \
        shell

# Start with empty source file list and add files for each directory
#

SRC_FILES =

#-----------------------------------------------------------------------
# Files for ../system                           #
#-----------------------------------------------------------------------

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    \
    ptinit.c    ptsend.c    ptrecv.c    ptcreate.c

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

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

#-----------------------------------------------------------------------
# Files for ../device/tty                       #
#-----------------------------------------------------------------------

TTY_SFILES =                                \
    ttyDispatch.S


TTY_CFILES =                                \
    ttyControl.c    ttyGetc.c   ttyInit.c   ttyInter_in.c   \
    ttyInter_out.c  ttyInterrupt.c  ttyKickOut.c    ttyPutc.c   \
    ttyRead.c   ttyWrite.c
```

Single function added to prototypes.h (to get the message passing function in our shell command going):

C:\Users\kmartin\Desktop\Pi\Classes\CPS675 OS\vbox\HW6\include\prototypes.h - Notepad++

File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?

prototypes.h | ptinit.c | ptsend.c | ptrecv.c | ptcreate.c | Makefile | xsh_create.c | main.c | ports.h

```
extern void msgPass1(void);


extern  void    runforever(void);

/* in file addargs.c */
extern  status  addargs(pid32, int32, int32[], int32,char *, void *);

/* in file ascdate.c */
extern  status  ascdate(uint32, char *);

/* in file bufinit.c */
extern  status  bufinit(void);

/* in file chprio.c */
extern  pri16   chprio(pid32, pri16);

/* in file clkupdate.S */
extern  uint32  clkcount(void);

/* in file clkhandler.c */
extern  interrupt clkhandler(void);

/* in file clkinit.c */
extern  void    clkinit(void);

/* in file clkint.S */
extern  void    clkint(void);

/* in file close.c */
extern  syscall close(did32);

/* in file control.c */
extern  syscall control(did32, int32, int32, int32);

/* in file create.c */
extern  pid32   create(void *, uint32, pri16, char *, uint32, ...);

/* in file ctxsw.S */
extern  void    ctxsw(void *, void *);

/* in file dot2ip.c */
extern  uint32  dot2ip(char *, uint32 *);

/* in file queue.c */
extern  pid32   enqueue(pid32, qid16);

/* in file intutils.S */
extern  intmask disable(void);

/* in file intutils.S */
extern  void    enable(void);

/* in file evec.c */
extern  int32   initevec(void);
```

Finally, the shell command xsh_create.c (we did not have to modify main.c)

C:\Users\kmartin\Desktop\Pi\Classes\CPS675 OS\vbox\HW6\shell\xsh_create.c - Notepad++

File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?

prototypes.h | ptinit.c | ptsend.c | ptrecv.c | ptcreate.c | Makefile | process.h | xsh_create.c | main.c

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

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


umsg32 msg, msg2;
/*------------------------------------------------------------------------
 * xsh_create - shell command to createeeeee
 *------------------------------------------------------------------------
 */
shellcmd xsh_create(int nargs, char *args[])
{
    int32   portNum, msgNum, portID;
    char    ch, ch1;
    char    *portCount, *msgCount;
    pid32   pid1, pid2;


    // if no arguments given, default to 5 ports 10 messages
    portNum=5;
    if (nargs == 1) {
        msgNum=10;

    }

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

        /*
```

C:\Users\kmartin\Desktop\Pi\Classes\CPS675 OS\vbox\HW6\shell\xsh_create.c - Notepad++

File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?

prototypes.h | ptinit.c | ptsend.c | ptrecv.c | ptcreate.c | Makefile | process.h | xsh_create.c | main.c | ports.h

```c
    }
    else {
        kprintf("Too many arguments\n");
        return 1;
    }


int32 portSend(int32 portID, umsg32 msg)
{
    ptsend(portID, msg);
    return OK;
}

int32 portRec(int32 msgNum, int32 portNum){
    int k, c;
    if(portNum>=msgNum){
        for(k=1;k<=msgNum;k++){
        msg2 = ptrecv(0);
        kprintf("Message at receiver: %d\n",msg2);
        }
    }else{
        for(c=1;c<=portNum;c++){
        msg2 = ptrecv(0);
        kprintf("Message at receiver: %d\n",msg2);
        }
        kprintf("Error! %d messages not received\n",msgNum-portNum);
        return 1;
    }

    return(OK);

}



void msgPass1()
{
    ptinit(portNum);
    ptcreate(portNum);
    int j;
    pid1 = create(portRec,1024,20,"Receiver",2,msgNum,portNum);
    resume(pid1);
    recvclr();
    int32 result = 1;
    for(j = 0;j<=msgNum;j++){
    result = portSend(0, j);
    }
    // sleepms(25);
    // if(result == OK)
        // kprintf("Message successfully passed!\n");
    // else
        // kprintf("Message pass failed! result\n");
}


    msgPass1();
    return 0;
```