

COP 4600 - Operating Systems Design
Spring 2015
Professor Sumi Helal

Lab 2: Extended Message Passing in Xinu

Due: 11:59 PM, February 22, 2015

I. Objectives

- You will learn how processes communicate with each other by passing messages, and understand potential issues during communication.
- You will learn how to create new message passing system calls, by extending the Xinu messaging interface to allow two special cases: (a) multiple messages to be sent to a single process, and (b) a message to be sent to multiple processes.

II. Background - communicating by message passing

Processes can communicate with each other by sending and receiving messages. Sender process creates a message and sends it to a specific receiver process. The receiver receives the message and then waits until the next message comes from any senders. In order to implement message passing, the Xinu process table is used; a flag (process has message) is used in addition to storage in the table to store the message itself. For message passing, Xinu provides a send/receive set of system calls as follows:

- send - send a message to a process
- receive – receive a message from some other process
- recvclr - check if a message has been received from another process and return it if it exists, but do not wait if one does not exist. Recvclr removes (clears) the message from the process table if one exists.
- recvtime - same as receive, yet will only wait for a limited specified time for the message to be received.

III. Extending send/receive system calls

In this lab, you will extend Xinu message passing capabilities and will implement your own system calls for handling two capabilities. For the first, your system call will allow sending multiple messages to a single process. In the second, a sender process will be able to send a message to multiple receivers.

A. System calls for sending/receiving multiple messages

A sender process may need to send multiple messages to a single process. For instance, different types of messages need to be sent at the same time. Or a message should be segmented in a fixed size and then sent to a receiver process. The messages should be received in the order in which they were sent. You may use a circular buffer for queuing (or

dequeuing) the messages as you did in Lab 1. You will be implementing the following system calls:

syscall sendMsg (pid32 <i>pid</i> , umsg32 <i>msg</i>);	Sending message (<i>msg</i>) to process (<i>pid</i>). In case a message or a number of messages are already waiting on <i>pid</i> to receive them, the new <i>msg</i> will be queued. It will return OK on success or SYSERR on error.
umsg32 receiveMsg (<i>void</i>);	Receiving a message from any sender. It causes the calling process to wait for a message to be sent (by any process). When a message is sent, it is received and returned.
uint32 sendMsgs (pid32 <i>pid</i> , umsg32* <i>msgs</i> , uint32 <i>msg_count</i>);	Sending a group (<i>msg_count</i>) of messages (<i>msgs</i>) to process (<i>pid</i>). It will return the number of <i>msgs</i> successfully sent or SYSERR on error.
syscall receiveMsgs (umsg32* <i>msgs</i> , uint32 <i>msg_count</i>);	Receiving a group (<i>msg_count</i>) of messages (<i>msgs</i>). It causes the calling process to wait for <i>msg_count</i> messages to be sent. When all messages are in the queue, they are then all together immediately received.

B. System call for sending to multiple recipients

Sometimes a sender process is required to send a message to multiple processes. For instance, after a process changes data, it should update the change in other processes which use the data. You will be implementing the following system calls;

uint32 sendnMsg (uint32 <i>pid_count</i> , pid32* <i>pids</i> , umsg32 <i>msg</i>);	Sending the message (<i>msg</i>) to the number (<i>pid_count</i>) of processes (<i>pids</i>). It will return the number of <i>pids</i> successfully sent to or SYSERR on error.
--	---

C. Requirements and notes

1. You need to add your new system calls in Xinu by updating include/prototypes.h. Insert the followings:

```
extern syscall sendMsg (pid32, umsg32);
extern uint32 sendMsgs (pid32, umsg32*, uint32);
extern uint32 sendnMsg (uint32, pid32*, umsg32,);
extern umsg32 receiveMsg (void);
extern syscall receiveMsgs (umsg32*, unit32);
```
2. Do not change existing system calls such as send, receive, recvclr, and recvtime, because they are used for other system processes. Also do not change the process

table entry in any way that causes existing system calls to no longer work. If necessary, make your own changes to the process stable (see include/process.h) by first commenting out the original table and make changes to a copy of the original. You need to be able to restore original Xinu after this assignment is completed so you can perform other Lab assignments.

3. The maximum number of messages is 10. And make the number of recipient processes no more than 3 for each of the senders and receivers (3 senders and 3 receivers).
4. Processes should print message(s) sent or received while the program is running. Senders should show recipients' pid and msgs, and how many messages successfully sent. In receivers' side, received messages should be displayed. Make sure you indicate in the print message whether a message is being sent or has been received.
5. You are able to add more functions and internal calls if necessary, as long as you never change existing functions.

IV. Submission

You should submit your *main()* program that would include all new system calls via SAKAI. Nothing else in the Xinu code should be changed. The TAs will replace Xinu's main by your main, and will compile and test, in addition of course to inspecting the code itself.