CS 354 Fall 2025

Lab 4: Custom Message Passing [20 pts]

Due: 9/29/2025 (Monday), 11:59 PM

1. Objectives

The objective of this lab assignment is to familiarize you with the steps involved in implementing message passing in XINU.

---

2. Readings:

Chapter 8 from the XINU textbook.

---

3. Source code

Start from a fresh copy of XINU, xinu-fall2025.tar.gz.

4. Custom Message Passing

(a) *mysend(), myreceive(), and myrecvclr()*

Xinu message passing (i.e., functions send(), receive(), and recvclr() implement process-to-process message passing with first-message semantics. Code the following new XINU system calls and place them in new files under `system/mysend.c`, `system/myreceive.c`, and `system/myrecvclr.c`:

```
syscall mysend(pid32 pid, int32 msg);

int32 myreceive(void);

int32 myrecvclr(void);
```

The new implementation must satisfy the following requirements:

1. Use messages of type `int32`.
2. Use last-message semantics, e.g., if two messages are successfully sent to a process before it calls `myreceive` the second message will be returned.
3. Unlike the base XINU implementation, do not make use of `procent` to store the message. You may however create flags in `procent`.
4. Perform the same error checking as the base implementations.
5. Use the value "`-1`" to indicate "no message."
6. `myreceive` must be synchronous (just like `receive`) and `mysend` must be synchronous (just like `send`)
7. The implementation does not need to implement timed waits.
8. The implementation must not interfere with the rest of the lab and XINU `send`/`receive`/`recvclr` implementation. For example, the implementation of `mysend` should not affect the functionality of `send`, `receive`, `recvclr`, `mysend2`, `receive2`, `recvclr2`,`vsend`, `vreceive`, and `vinit`.

(b) *mysend2(), myreceive2(), and myrecvclr2()*

Code the following new XINU system calls and place them in new files under `system/mysend2.c`, `system/myreceive2.c`, and `system/myrecvclr2.c`:

```
syscall mysend2(pid32 pid, int32 msg);

int32 myreceive2(void);

int32 myrecvclr2(void);
```

The behavior of the `mysend2`/`myrecvclr2` will be identical to `mysend`/`myrecvclr`. However, `myrecv2` does not block when there is no message and returns "no message" (`-1`) appropriately. The implementation must not interfere with the rest of the lab and XINU `send`/`receive`/`recvclr` implementation. For example, the implementation of `mysend2` should not affect the functionality of `send`, `receive`, `recvclr`, `mysend`, `receive`, `recvclr`,`vsend`, `vreceive`, and `vinit`.

5. Variable Length Message Passing

Imagine an entirely new message passing system for XINU which supports variable-length messages. a given message can be between 1 and K bytes, where K is specified when the mechanism is initialized. Furthermore, the mechanism restricts the total outstanding (unreceived) messages across all processes to no more than N, where N is specified when the mechanism is initialized. The idea is to limit total memory used for message passing. (Note that N can be less than NPROC). To achieve this, code the following new XINU system calls and place them in new files under `system/vinit.c`, `system/vsend.c`, and `system/vreceive.c`:

```
syscall vinit(uint32 maxmsglen, uint32 maxoutstanding);

syscall vreceive(char *messagebuf, int32 maxlen);

syscall vsend(pid32 process_id, char *message, int32 msglength);
```

The new implementation must satisfy the following requirements:

1. `vinit` must be called before `vreceive`/`vsend` and sets the maximum values of the variable messaging system. If `vinit` was not called, `vsend` and `vreceive` should fail.
2. A message's minimum and maximum length is 1 and 60 bytes respectively
3. The maximum number of outstanding (unreceived) messages for the whole system is `NPROC`
4. Like the original XINU `send`/`receive`, use first message semantics
5. There is no need to implement timed wait
6. `vinit`/`vreceive`/`vsend` return SYSERR on failure
7. `vreceive`/`vsend` return the number of bytes successfully received and sent while `vinit` returns `OK` on success
8. `vsend` expects the message passed in to contain a message of `msglength` in memory.
9. `vsend` blocks if sending the message exceeds the `maxoutstanding` value `vinit` was passed e.g. if `vsend` is attempting to send the `NPROC+1`th message it will block if `vinit` was initialized with `maxoutstanding = NPROC`
10. `vreceive` blocks until a message arrives
11. `vreceive` expects the `messagebuf` passed into have access to the `maxlen` bytes in memory.

12. `vreceive` reads the minimum of remaining message length, `maxlen` and `maxmsglen` (value `vinit` was called with).
13. The implementation must not interfere with the rest of the lab and XINU `send`/`receive`/`recvclr` implementation. For example, the implementation of `vsend` should not affect the functionality of `send`, `receive`, `recvclr`, `mysend2`, `receive2`, `recvclr2`,`mysend`, my`receive`, and `myrecvclr`.

Important: When new functions including system calls are added to XINU, make sure to add its function prototype to include/prototypes.h. The header file prototypes.h is included in the aggregate header file xinu.h. Every time you make a change to XINU, be it operating system code (e.g., system call or internal kernel function) or app code, you need to recompile XINU on a frontend Linux machine and load a backend with the new xinu.xbin binary. To ensure your new system calls are compiled please use "make rebuild".

---

6. Questions

Place **short answers** to the following questions in a file named `questions.pdf`. In the case of using document software (e.g., Word), please make sure to convert the file to **PDF**.

1. What is the minimum amount of messages required when using `mysend` to send the string "hello\n" (the quotes are not sent)?
2. Is this code nondeterministic?

```
void process() {
    int a;
    vsend(getpid(), (char *)&a, sizeof(int));
}
```

3. Suppose `vinit(1,1)` was called and NPROC is 1. What is the return value?
4. For the variable message size implementation, how many reads will a receiver process require to read a 60 byte message if the buffer used is 7 bytes?
5. How would you modify the system without changing `vsend/vreceive` such that the messages sent notify the receiver of who sent the message? (Hint: Design a struct)

Turn-in instructions

General instructions:

When implementing code in the labs, please maintain separate versions/copies of code so that mistakes such as unintentional overwriting or deletion of code is prevented. This is in addition to the efficiency that such organization provides. You may use any number of version control systems such as GIT and RCS. Please make sure that your code is protected from public access. For example, when using GIT, use git that manages code locally instead of its on-line counterpart github. If you prefer not to use version control tools, you may just use manual copy to keep track of different versions required for development and testing. More vigilance and discipline may be required when doing so.

Submission format for Lab 4:

```
lab4

|--- xinu-fall2025

|------ system

|--------- send.c

|--------- vsend.c

...

|------ include

|--------- prototypes.h

...

|--- questions.pdf
```

Electronic turn-in instructions:

1) Please make sure to follow the file and function naming convention as mentioned in the lab handout.
2) Please make sure your code does not depend on the contents of `main.c`, we will replace this during testing

3) Go to the `xinu-fall2025/compile` directory and run: `make clean`.
4) Go to the directory where `lab4` (lab directory containing `xinu-fall2025/` and `questions.pdf`) is a subdirectory.
    a) For example, if `/homes/alice/my_cs354_stuff/lab4/xinu-fall2025` is your directory structure use the following commands to submit your lab
5) Type the following command

```
cd /homes/alice/my_cs354_stuff
turnin -c cs354 -p lab4 lab4
```

You can check/list the submitted files using

```
turnin -c cs354 -p lab4 -v
```

Please make sure to disable all debugging output before submitting your code.