Project 1A COM S 352 Spring 2024

(Due: Thursday, Feb 8, 11:59pm)

This project is to be completed by yourself. You may ask classmates for help installing software, setting up your development environment, and understanding the sample program.

1. Preparation

As a pre-requisite of this project, you should have set up the xv6-riscv system. Refer to the document titled "Introduction to xv6 (part I)" and "VS Code configuration to use pyrite" on Canvas for more instructions on system installation. In addition, if you prefer to install the system on your own native/virtual Linux/macOS machine, refer to https://pdos.csail.mit.edu/6.828/2019/tools.html.

2. Test and read an example

Attached is a sample program, name "broadcast.c", in which a parent process creates a certain number of child processes, sends a message to all of these child processes through a pipe, and receives the acknowledgement of completion through another pipe.

2.1 Add the program to your xv6-riscv system

Download the program file "broadcast.c" and save it to the directory of xv6-riscv/user. Then, add "_broadcast" to the end of UPROGS in Makefile in directory xv6-riscv/. That is, UPROGS in Makefile should be as follows:

```
…
UPROGS = \
        … …
        $U/_zombine\
        $U/_broadcast\
…
```

2.2 Set number of CPUs emulated

In Makefile, also change the value of CPUS to 1 to simplify scheduling and make outputs more intelligible.

CPUS :=1

2.3 Run the program in the shell

Type “make qemu” to compile the programs and runs the shell.

In the shell, type “broadcast <number_of_receivers> <msg_to_broadcast>” to start the program. For example, to create 5 receivers and broadcast message “Hello!” to all of them, type:

broadcast 5 Hello!

You will get the output as follows:

Parent: creates child process with id: 0
Child 0: start!
Parent: creates child process with id: 1
Child 1: start!
Parent: creates child process with id: 2
Child 2: start!
Parent: creates child process with id: 3
Child 3: start!
Parent: creates child process with id: 4
Child 4: start!
Parent broadcasts: Hello!
Child 0: get msg (Hello!)
Child 1: get msg (Hello!)
Child 2: get msg (Hello!)
Child 3: get msg (Hello!)
Child 4: get msg (Hello!)
Parent receives: completed!

2.3 Read program: broadcast.c

Read the program to understand how it works.

In the main function, system call “pipe” is made to have two pipes created, and system call “fork” is made for several times to create child processes. Then, the parent and child processes work differently as follows.

- The parent process’ operations:

    - Broadcast message – the parent process fills the message's flags field with 1s to indicate that all the receivers need to receive the message, and fills the message's content field with the message content. After that, it writes the message to the pipe named "channelToReceivers".
    - Receive acknowledge – the parent calls read operation on the pipe named "channelFromReceivers" to wait for the broadcast to complete. Note: when a pipe is empty, the process who calls read on it is blocked till the pipe is filled with some content.
    - End the process.
- Each child process' operations:
    - Announce the start of the child process.
    - Read the pipe named "channelToReceivers" to get the message.
    - Check if all receivers have already received this message (I.e., the flags for all receivers have been set 0). If so, send acknowledgement to the parent process through the pipe named "channelFromReceivers"; otherwise, the received message is written back to the pipe named "channelToReceivers" so that the child processes yet to receive the message can receive it. Note: content in a pipe is consumed (removed) when the pipe is read.
    - End the process

Pay attention to how the system calls are used in the program.

3. Develop a new program: unicast.c

Now you are required to develop a new program named "unicast.c", by revising program "broadcast.c".

Program "unicast.c" should take three arguments: the number of child processes, the id of the receiver, and the message to send to the receiver. For example, suppose the number of child process is 5, the id of the receiver is 3 (I.e., the fourth child), and the message to send is "Hello!". Then, the program is run by typing the following:

```
unicast 5 3 Hello!
```

The above launching of unicast would produce the following output:
Parent: creates child process with id: 0
Child 0: start!
Parent: creates child process with id: 1
Child 1: start!
Parent: creates child process with id: 2
Child 2: start!

Parent: creates child process with id: 3
Child 3: start!
Parent: creates child process with id: 4
Child 4: start!
Parent sends to Child 3: Hello!
Child 0: get msg (Hello!) to Child 3
Child 0: the msg is not for me.
Child 0: write the msg back to pipe.
Child 1: get msg (Hello!) to Child 3
Child 1: the msg is not for me.
Child 1: write the msg back to pipe.
Child 2: get msg (Hello!) to Child 3
Child 2: the msg is not for me.
Child 2: write the msg back to pipe.
Child 3: get msg (Hello!) to Child 3
Child 3: the msg is for me.
Child 3 acknowledges to Parent: received!
Parent receives: received!

Specifically, the program should implement the following:

- The parent process should create a specified number of child processes and each child process should declare its start (as in broadcast.c).
- The parent process should then send the specified message to the specified receiver through a pipe, which you can reuse the pipe named “channelToReceivers” in broadcast.c. (Hint: the message now should have a field specifying who is the intended receiver). Then, it waits to receive acknowledgement from another pipie, which you can reuse the one named “channelFromReceivers” in broadcast.c.
- Each child process should read the pipe to get the message, and check if the message is for it. If yes, the child process should send the acknowledge of “received” to the parent through the pipe named “channelFromReceivers”; otherwise, it should write the message back to the pipe so that the intended receiver child process can eventually receive this message. For either case, the child process terminates afterwards.
- When the parent process receives acknowledgement of “received”, it terminates.

You are free to copy the code from “broadcast.c” when develop “unicast.c”. You should add “_unicast” to Makefile, so that it can be run in the shell after the new system is compiled.

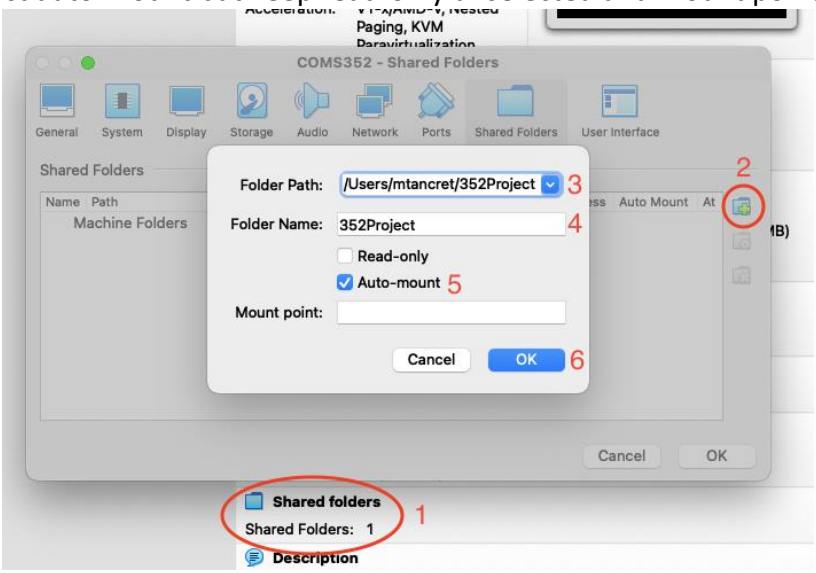
4. Submission

You are required to submit two files: “unicast.c” and Makefile. Add comments to the c program to help grader understand your code. Make sure the program compiles without any errors.

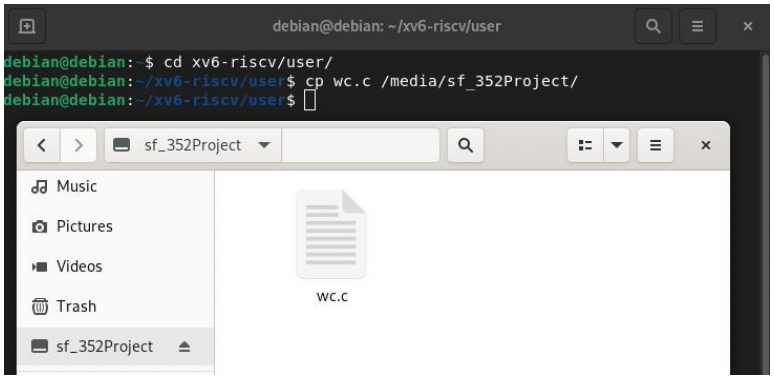
5. Notes: How to Copy Files out of the Virtual Machine

These instructions are assuming the use of VirtualBox, other virtualization software have a similar way to share files. Note: If you use pyrite, the files in xv6 can be accessed as normal files.

The simplest way to move files from the virtual machine to your host machine is by creating a shared folder. Shutdown the virtual machine if it is currently running, and on the main Manager window click on Shared Folders. Navigate to a Folder Path on your where you want to store project files, select auto-mount but keep read-only unselected and mount point empty.



After restarting the virtual machine you will be able to copy files into the directory `/media/sf_folder_name_you_chose`. Those files will be available at the folder path you chose on your host machine.


debian@debian: ~/xv6-riscv/user
debian@debian:~$ cd xv6-riscv/user/
debian@debian:~/xv6-riscv/user$ cp wc.c /media/sf_352Project/
debian@debian:~/xv6-riscv/user$
sf_352Project
Music
Pictures
Videos
Trash
sf_352Project
wc.c