COP 4600
Operating Systems Design
Spring 2015
Professor Sumi Helal

Lab 0: Getting to Know Xinu

- Nothing due here.
- Will be covered during Monday Jan 26, 2015 Recitation Classes

---

Objectives:

- Get familiar with a the virtual machine working environment
- Compile a copy of XINU from source on the front end and load it in into the backend working environment
- Modify the source of XINU to start several processes that print messages in the terminal.
- If you do not like (or do not know) vi which is the de facto editor on the developer machine, then you will also lean how to get a copy of the source shared through Virtual Box with your host machine. This way you can access the source on the host machine with more powerful IDE's such as Eclipse.

1. XINU configuration

If you have not already successfully followed the "How to build XINU on VirtualBox" tutorial, its time to make it run. Make sure you can login to XINU. Remember the images of the virtual machines (Appliances) can be downloaded from the following website:

ftp://ftp.cs.purdue.edu/pub/comer/private/Xinu/

2. Using the XINU shell

By default, when XINU boots up, it runs a shell (xsh). To view all the usable shell commands, run the help command

```
xsh $ help

shell commands are:
```

```
argecho     date        exit        led         ping            udpecho   arp
devdump     help        memdump     ps          udpeserver        cat
echo        ipaddr      memstat     sleep       uptime          clear
ethstat     kill        nvram       udpdump     ?
```

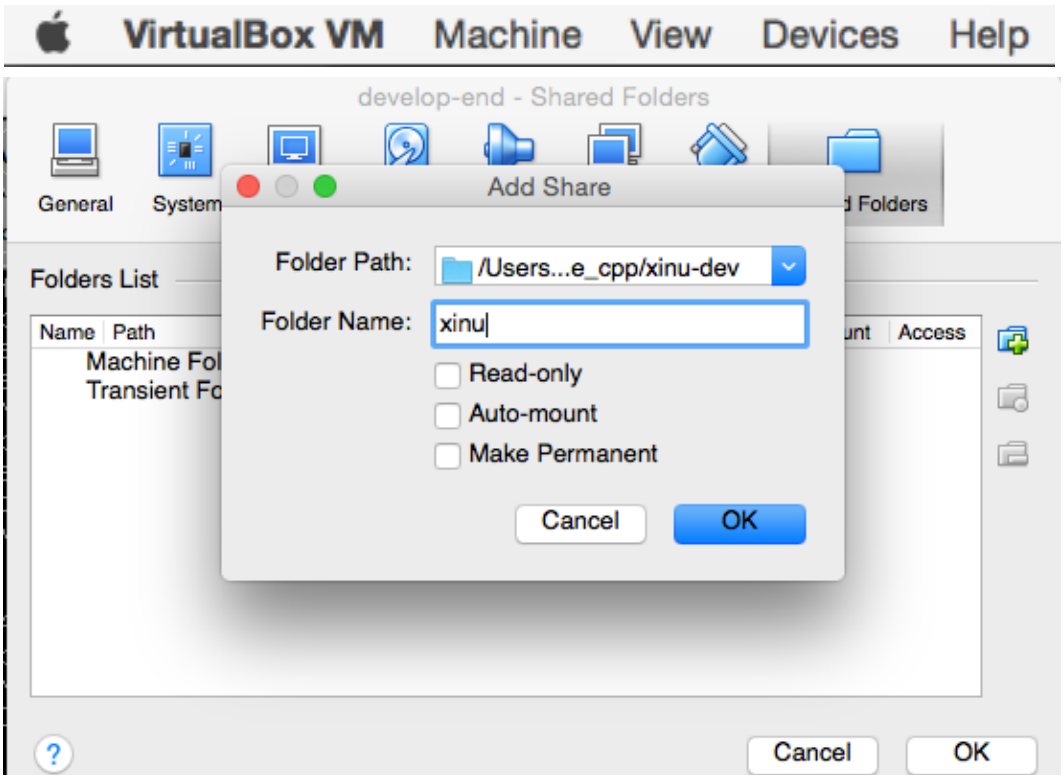
Take a moment to run some shell commands and view their output. Specifically make sure you run the ps command which lists information about running processes.

Here you can see the several pieces of information for all processes currently running. The name gives some detail about what the process is intended for. XINU always contains a special process with process identifier (pid) 0 called the null process (or prnull). This process is the first process created by XINU and is always ready to execute. The inclusion of this process ensures that there is always something for XINU to execute even if all other processes have finished or are waiting for something. It behaves similar to the "System Idle Process" in the Windows Operating System.

3. Setting up the code on Eclipse

In order to be able to modify the source code of XINU, you may do so on the developer machine using vi. If you want to use other more favorite text editors or IDE's you have to share the code with your host machine. In this lab 0, I will load the code onto eclipse to make it easier to modify.

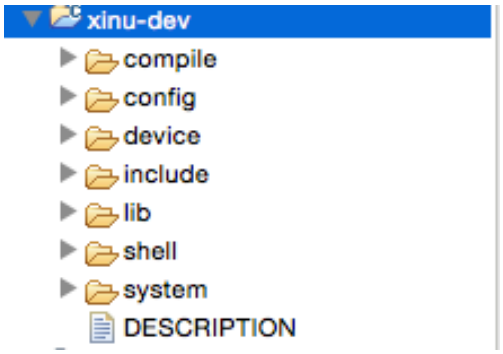
First, we need to grab the source code from the virtual machine and deploy it on the host machine. For this purpose we are going to create a shared folder between our develop-end virtual machine and host machine. On your host machine create a folder "xinu-dev". Start the develop-end machine and go to **devices>shared folder settings>add new** on folder path select "other" and link to the xinu-dev empty folder and change folder name textbox to "xinu" (VirtualBox doesn't like special characters).



On the command line in the develop-end machine create a new folder "xinu-dev" with the command **mkdir xinu-dev** mount the shared folder on it using the command **sudo mount –t vboxsf xinu /home/xinu/xinu-dev/** and copy the source code with the command **sudo cp –r /home/xinu xinu-x86-vm/* /home/xinu/xinu-dev**

Note: after you shut down the virtual machine you will only need to link the shared folder on the devices menu and issue the mount command.

Now, import xinu-dev folder into eclipse as an existing makefile project. To be able to perform this step you need to have an eclipse version that supports C environment. You can have an eclipse instance exclusive for C development or download the C plugin for your existing eclipse environment. Only the IDE is needed since we will be compiling on the development virtual machine. Your project should look like this.



Now you are ready to modify XINU!

4. Creating a process in XINU

The code for the first user process that is created by XINU is located in system/main.c. The code by default creates a single process to execute the XINU shell. The code to create she shell process looks like this:

```
resume(create(shell, 4096, 1, "shell", 1, CONSOLE));
retval = recvclr();
while (TRUE) {
        retval = receive();
        kprintf("\n\n\rMain process recreating shell\n\n\r");
        resume(create(shell, 4096, 1, "shell", 1, CONSOLE));
}
```

Processes in XINU are created using the create system call which does the following:

1. Creates all necessary control structures for the new process
2. Initializes the process stack for the new process

Note: you can cmd+click (Mac) on the create function to see the source code in eclipse.

The call to create in main.c creates a new process to execute the code located in the "shell" function.

1. The stack size is set to 4096 bytes
2. The priority is set to 1 (the larger the number the higher the priority)
3. The name of the process is set to "shell"
4. There is 1 argument sent to the shell function: the value of CONSOLE

NOTE: when a process is created, it is not automatically executed. The status of the process must be changed to "ready". This is done by using the resume system call which takes as input the processed identifier. This tells XINU that the processes is ready to execute on the CPU.

Create the following function at the top of main.c:

```
void myprocess(int a) {
        kprintf("Hello world %d\n", a);
}
```

Comment the code in main.c that creates a shell process and replace it with the following:

```
resume(create(myprocess, 4096, 50, "helloworld", 1, 1));
while(TRUE) {
        // Do nothing
}
```

To make your modification run, with the shared folder mounted, move to the compile folder using the command **cd xinu-dev/compile** and then run the following commands **make clean; sudo make; ./upload.sh** and start the back-end virtual machine.

Once XINU boots up, you should see the message "Hello world" printed to the console.

Add some additional calls to create and resume to create several processes that call the myprocess function.

```
resume(create(myprocess, 4096, 50, "helloworld1", 1, 1));
resume(create(myprocess, 4096, 50, "helloworld2", 1, 2));
resume(create(myprocess, 4096, 50, "helloworld3", 1, 3));
resume(create(myprocess, 4096, 50, "helloworld4", 1, 4));
resume(create(myprocess, 4096, 50, "helloworld5", 1, 5));
while(TRUE) {          // Do nothing  }
```

Compile, load, and run this new code. You should see multiple messages printed to the console (one for each process).

5. Changing process priority

The process scheduler in XINU uses a very simple rule for deciding which process to run. Each process gets a set time to run on the CPU before it gets context switched for another process. XINU always chooses to execute the highest priority process that is ready to run.

Create another function in the main.c file that contains the following code:

```
void myhungryprocess(void) {
        kprintf("I'm a looping process\n");
        while(TRUE) {
                // Do nothing forever
        }
}
```

Now modify your existing code in the main function to call the new process, but give it a high priority:

```
resume(create(myhungryprocess, 4096, 1000, "hungryprocess", 0));
resume(create(myprocess, 4096, 50, "helloworld1", 1, 1));
resume(create(myprocess, 4096, 50, "helloworld2", 1, 2));
resume(create(myprocess, 4096, 50, "helloworld3", 1, 3));
resume(create(myprocess, 4096, 50, "helloworld4", 1, 4));
while(TRUE) {          // Do nothing  }
```

What do you observe when you run this on a XINU backend?

Since XINU always chooses the highest priority processes to run and all 5 of the processes created are ready, the hungry process with priority 1000 gets to execute on the CPU. From that point on, every time the XINU scheduler runs, it chooses the hungry process to run and no other processes get to execute.

Furthermore, take a look at initialize.c. This file contains the code that creates the process to run the main function. What priority is main created with? Given main's priority in the example above does main get to run after hungryprocess is created?

Familiarize yourself with setting priorities of various processes. Take a look at the output from the ps command to see the priorities for the processes created by XINU at startup.