CS 354 - Spring 2025

XINU Set-up

Objectives:

The following instructions describe how to:

- Log onto a XINU Lab frontend PC by remote access or physically going to the XINU Lab. Download, extract, and compile a copy of the XINU source.
- View the status of existing XINU backends.
- Load a built XINU image onto a XINU backend and successfully boot the image.
- Modify the XINU source code initialize.c which creates the first XINU app process that executes function main() which is located in system/main.c.

---

1. XINU Configuration

To use XINU, several environment variables must be set. First log onto one of the frontends **xinu01.cs.purdue.edu**, **xinu02.cs.purdue.edu,** ..., **xinu21.cs.purdue.edu** which are Linux PCs. For remote login see

Section 5. To login you should be able to use your career account user and password. Please note that CS has recently moved to two-factor authentication when using ssh. If you have any trouble with your account, please contact ScienceHelp@purdue.edu.

Note: If you are accessing a frontend machine remotely from a Windows machine, it is strongly recommended that you install and use the freeware PuTTY. Using Windows PowerShell or cmd terminal may not correctly map keyboard input CTRL-space correctly which is needed to connect to a backend machine from a frontend machine. The default configuration of PuTTY on Windows 10 and 11 (and likely earlier versions) will work correctly. Another stable working environment is to install WSL2 (Windows Subsystem for Linux version 2) on Windows running Ubuntu and open an Ubuntu terminal instead of PowerShell/cmd. If you choose not to use PuTTY or WSL2 on Windows, you are responsible for handling issues that may arise. There should be no issues when accessing the frontends remotely from a Linux or MacOS machine via ssh.

The following assumes that your shell is bash. The syntax may vary slightly if you use a different shell. (Run "echo $0" to determine your current shell.)

Setting environment variables for XINU:

1. Edit **.bashrc** in your home directory by adding /p/xinu/bin to your path:

   export PATH=${PATH}:/p/xinu/bin

2. Run **source .bashrc** (or its equivalent) to make the change take effect.

Accessing and untarring XINU tarball source files:

1. Change to your home directory, if you are not already in it.

2. Unpack:

   ```
   tar zxvf /homes/cs354/xinu-spring2025.tar.gz
   ```

   In your home directory, you will now have a directory called xinu-spring2025. The subdirectories under this directory contain source code, header files, configuration files, and Makefile for compiling XINU.

---

2. Building XINU

To compile the XINU kernel which will be loaded and executed on a backend machine, run "make" in the **compile/** directory:

```
% cd xinu-spring2025/compile
% make clean
% make
```

This creates a loadable executable file named **xinu.xbin**. This file is loaded on a backend machine in Section 3 ('Running XINU').

If, upon loading xinu.xbin and power cycling to run XINU the system hangs, run 'make rebuild' then 'make clean' followed by make to rebuild a XINU image that removes dependencies from the current build. In cases where a change is incremental and straightforward, just running 'make' may suffice to generate a correct build without slowing down gcc by recompiling the entire source code. If XINU hangs or crashes after following the above steps, the cause is likely to be bugs in your code modifications. To avoid starting from scratch, maintain working versions that you can return to.

3. Running XINU

The executable XINU binary runs on a selected backend machine. We have 96 dedicated backends: **galileo101.cs.purdue.edu**, ..., **galileo196.cs.purdue.edu**. As these are real physical machines, some may be down due to hardware and software issues. If they do not function as specified below, select another machine and try again.

The backends are all Intel Galileo development boards which run the quark system on chip processor.

The backend machines are shared resources. When a backend machine is grabbed by a student, it is dedicated for use by that student to run his/her version of XINU. To see which backends are available for booting XINU, type:

```
% cs-status -c quark
```

This will show you who is using each backend and how long they have been using it. As with all hardware, sometimes they fail and may become unavailable until repaired by our technical staff.

To boot your copy of XINU on a backend, connect to a back-end, say, galileo115, by issuing the command:

```
% cs-console galileo115
```

Similarly for the other backends by specifying a different backend name. With no arguments cs-console will connect you to the first available backend (including broken ones). Occasionally, backend hardware malfunctions. If you suspect this to be the case, please inform the TAs and select a different backend.

To load your copy of XINU onto a selected backend perform:

```
(control-@) OR (control-spacebar)        //esc to local command-
mode

(command-mode) d        //download command

file: xinu.xbin          //tell it to download 'xinu.xbin' (this
example assumes that you are in the xinu-spring2025/compile
directory)

(control-@) OR (control-spacebar)       //esc to local command-
mode

(command-mode) p        //power cycle the backend
```

After several seconds XINU should boot with a "Welcome to Xinu!" message that looks something like this:

```
Xinu for galileo -- version #18  (park)  Thu Jan  9 02:21:47 PM EST
2025

Ethernet Link is Up
MAC address is 98:4f:ee:00:09:3d
 250088992 bytes of free memory.  Free list:
```

```
          [0x001591E0 to 0x0EFD8FFF]
          [0x0FDEF000 to 0x0FDEFFFF]
    103629 bytes of Xinu code.
          [0x00100000 to 0x001194CC]
    132840 bytes of data.
          [0x0011CBC0 to 0x0013D2A7]


Hello World!

I'm the first XINU app and running function main() in system/main.c.

I was created by nulluser() in system/initialize.c using create().

My creator will turn itself into the do-nothing null process.

I will create a second XINU app that runs shell() in shell/shell.c as
an example.

You can do something else, or do nothing; it's completely up to you.

...creating a shell


------------------------------------------

   __    __   _____    _   _    _    _
   \ \  / /  |__ __|  | \ | |  | |  | |
    \ \/ /     | |    |  \| |  | |  | |
    / /\ \    _| |_   | \   |  | |  | |
   / /  \ \  |     |  | | \ |  \  --  /
   --    --   -----    -   -     ----
------------------------------------------


Welcome to Xinu!


xsh $
```

To disconnect and free up the backend:

```
(control-@) OR (control-spacebar)
```

```
(command-mode) q        //quit
```

If there are any issues disconnecting please power cycle before quitting:

```
(command-mode) p        //power cycle the backend; wait until you see
"Press [Enter] ..."

(control-@) OR (control-spacebar)

(command-mode) q        //quit
```

NOTE:

Please do not leave a running copy of your XINU on a backend. This may prevent others from using that backend.

---

4. Troubleshooting

1. We only show the mapping from bash to tcsh. If you use other shell types, please contact the TAs. The mapping is as follows: Edit **.cshrc** instead of **.bashrc**. Add a line **setenv PATH ${PATH}:/p/xinu/bin** instead of **export PATH=${PATH}:/p/xinu/bin**. Run **tcsh** instead of **source .bashrc**.
2. Try to figure out what's going on by yourself. Oftentimes the steps described above were not precisely followed.

3. When your XINU executable misbehaves or crashes (i.e., does not do what you intended when programming the kernel) then debug the problem(s) and try again. Since your version of the XINU kernel runs over dedicated hardware, you are in full control. That is, there are no hidden side effects introduced by other software layers that you are not privy to. By the same token, everything rests on your shoulders.
4. If you get repeatedly stuck with "Booting XINU on ... " please contact the TAs.
5. If you are not able to get a free backend, please contact the TAs.

---

5. Remote Login

You can remote access the frontend lab PCs using TLS/SSL applications such as ssh on Linux/MacOS and ssh.exe using Command/Power shell (or PuTTY, OpenSSH, etc.) on Windows. Please note that CS has recently moved to two-factor authentication when using ssh. Use of the backends is limited to implementing, testing, and evaluating lab assignments of CS

354. You access the backends through one of the frontend machines in the XINU Lab.

---

6. An Example App: 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         help         netinfo
udp          ?
        arp          devdump      kill         ping         udpecho
        cat          echo         memdump      ps
udpeserver
        clear        exit         memstat      sleep        uptime
```

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.

```
        xsh $ ps
        Pid Name             State Prio Ppid Stack Base Stack Ptr
Stack Size
        --- ---------------- ----- ---- ---- ---------- ---------- --
--------
          0 prnull           ready    0    0 0x0EFDEFFC 0x0EFDEEC0
8192
          1 rdsproc          wait   200    0 0x0EFDCFFC 0x0EFDCAAC
16384
          2 Main process     recv    20    2 0x0EFD8FFC 0x0EFD8F64
65536
```

```
          3 shell            recv     50     3 0x0EFC8FFC 0x0EFC8C6C
8192
          4 ps               curr     20     4 0x0EFC6FFC 0x0EFC6E18
8192
```

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. Unlike other processes, prnull is handcrafted without using create(). 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. By default, UNIX/Linux and Windows have similar null or idle processes.

Back to the CS 354 web page