CS 444/544 OS II
Lab Tutorial #1
Lab Setup, Tools, and Debugging

Acknowledgement: Slides drawn heavily from Yeongjin Jiang
How Do We Run Lab Sessions?

Tutorial Video (30 ~ 45 minutes)

Follow the instructions (slides/video)

Exercise + Q&A

Do your lab exercises and ask questions to TAs (on Discord)

Lab instruction website:
https://canvas.oregonstate.edu/courses/1912331/pages/labs
Lab Instructions

**Getting Started with x86 assembly**

If you are not already familiar with x86 assembly language, you will quickly become familiar with it during this course! The [PC Assembly Language Book](#) is an excellent place to start. Hopefully, the book contains mixture of new and old material for you.

*Warning:* Unfortunately the examples in the book are written for the NASM assembler, whereas we will be using the GNU assembler. NASM uses the so-called *Intel* syntax while GNU uses the AT&T syntax. While semantically equivalent, an assembly file will differ quite a lot, at least superficially, depending on which syntax is used. Luckily the conversion between the two is pretty simple, and is covered in [Brennan's Guide to Inline Assembly](#).

---

**Note**

**Exercise 1.** Familiarize yourself with the assembly language materials available on the [cs444 reference page](#). You don't have to read them now, but you'll almost certainly want to refer to some of this material when reading and writing x86 assembly.

We do recommend reading the section “The Syntax” in [Brennan's Guide to Inline Assembly](#). It gives a good (and quite brief) description of the AT&T assembly syntax we'll be using with the GNU assembler in JOS.

Certainly the definitive reference for x86 assembly language programming is Intel's instruction set architecture reference, which you can find on the [cs444/544 reference page](#) in two flavors: an HTML edition of the old [80386 Programmer's Reference Manual](#), which is much shorter and easier to navigate than more recent manuals but describes all of the x86 processor features that we will make use of in cs444/544; and the full, latest and greatest [IA-32 Intel Architecture Software Developer's Manuals](#) from Intel, covering all the features of the most recent processors that we won't need in class but you may be interested in learning about. An equivalent (and often friendlier) set of manuals is available from [AMD](#). Save the Intel/AMD architecture manuals for later or use...
TA Availability – Lab Q&A (Discord)

- Course Discord: [https://discord.gg/kJMsmtbfF](https://discord.gg/kJMsmtbfF)
- Office hours: [https://canvas.oregonstate.edu/courses/1912331/](https://canvas.oregonstate.edu/courses/1912331/)
JOS Lab (lab1-lab4, 70%)

Lab 1: Booting (10%)

Learn how an OS boots from BIOS to bootloader to the OS kernel

Lab 2: Virtual Memory (15%)

Learn how to manage physical/virtual memory space in an OS kernel
JOS Lab (1-4, 70%)

Lab 3: User Environment and System Call (20%)

Learn how user/kernel execution switch works and providing an isolated virtual memory space to a user process

Lab 4: Preemptive Multitasking (25%)

Learn how user/kernel execution switch works and providing an isolated virtual memory space to a user process
Extra Credit Labs

JOS Challenges (1% each from Lab 1,2,3, same due with the lab)

Solving a challenge would add +1% towards the entire course credits

Note

Challenge (Extra credit 1%). Enhance the console to allow text to be printed in different colors. The traditional way to do this is to make it interpret ANSI escape sequences embedded in the text strings printed to the console, but you may use any mechanism you like. There is plenty of information on the cs444/544 reference page and elsewhere on the web on programming the VGA display hardware. If you’re feeling really adventurous, you could try switching the VGA hardware into a graphics mode and making the console draw text onto the graphical frame buffer.

To get 1% of credit, please create a command ‘show’ in the monitor and print a beautifule ASCII Art with 5 or more colors when the command is typed on the console.

Once you finish this, please create a file .lab1-extra at the root of your repository directory (under jos/). We will use that file as an indicator that you finished this extra-credit and then grade your work accordingly.
Today’s Tutorial

1. Lab environment setup
2. Commit your information on your own ‘jos’ repository
3. Run JOS with TMUX
**ACTION**: Setup the lab environment on OS servers

Connect to any of the following servers):

os2.engr.oregonstate.edu

oldos2.engr.oregonstate.edu

oldos1.engr.oregonstate.edu

os1.engr.oregonstate.edu

RUN (please copy-and-paste):

```
$ /nfs/farm/classes/eecs/spring2023/cs444-001/cs444-setup.py
```

This will setup BASH, VIM, GDB, QEMU, and Tmux
Running Script

Type ‘y’ if you wish to use the default setup...

```bash
os2 ~ 166% /nfs/farm/classes/eecs/spring2021/cs444-001/cs444-setup.py
Cloning into '/nfs/stak/users/songyip/.cs444'...
remote: Enumerating objects: 19, done.
remote: Counting objects: 100% (19/19), done.
remote: Compressing objects: 100% (11/11), done.
remote: Total 432 (delta 7), reused 15 (delta 5), pack-reused 413
Receiving objects: 100% (432/432), 10.98 MiB | 0 bytes/s, done.
Resolving deltas: 100% (259/259), done.

Do you want to install peda to ~/.gdbinit (y/n) ?
Do you want to install cs444 custom tmux configuration (y/n) ?
Do you want to install .bashrc (y/n) ?
Do you want to install .vimrc and vim plugins (y/n) ?
```
ACTION: Generate Public Key (Step 1)

If you already have your ssh public/private key on our servers, you can use the same public key.

Otherwise, please create one by typing the following command:

```
$ ssh-keygen -t ecdsa
```

After that, please print your public key, and then copy the key to the clipboard

```
$ cat ~/.ssh/id_ecdsa.pub
```

ecdsa-sha2-nistp256 (THIS IS A SAMPLE PUBLIC KEY)
AAAAAE2VjZH NhLXNoYTITbmlzdHAyNTYAAAAAIbmlzdHAyNTYAAABBFRx1q/fIouV7Kf1GVEwL04/yIp rKdtf9KYOHk8gAbtIxocFFsAgBuEzRg4EtjQEcYnitroSm2F14mHy2cz27+ho= songyip@os2.engr.oregonstate.edu
Generate Public/Private Key (Step 2)

Use your favorite command text editor (i.e., vim) to open up ~/.ssh/authorized_keys.

If it does not exist, creat it.

Paste the public key you copied in the last step

```
ecdsa-sha2-nistp256 (THIS IS A SAMPLE PUBLIC KEY)
AAAAAE2VjZHNhLXNoYTItbmlzdHAYNTYAAAAlbmlzdHAYNTYAAABBBFRx1lq/fIouV7Kf1GVEwL04/yIprKdt
f9KYOHK8gAbtIxocFFsAgBuEzRg4EtjQEfYnitroSm2F14mHy2cz27+ho=
songyip@os2.engr.oregonstate.edu
```

You now need to set permissions on the file. Type

```
chmod 600 ~/.ssh/authorized_keys
```
ACTION: https://github.com/
Register your account! (in case you don’t have one!!)

Visit the website and register an account;

Use @oregonstate.edu e-mail address (you can’t register otherwise)
**ACTION**: Cloning jos-lab repository (step 1)

Logon to github classroom using your github account.
https://classroom.github.com/classrooms

Accept the JOS assignment using the invitation link:
- [https://classroom.github.com/a/Sd0BVeHs](https://classroom.github.com/a/Sd0BVeHs)

Note: you need to link your email address to your github account for the first time

Clone the repository by running the following command on flip:
SSH: $ git clone git@github.com:OSU-OS2-S23/jos-labs-yourusername.git
**ACTION**: Cloning jos repository (step 2)

Note that this repo contains nothing but a setup script.

Since GitHub Classroom does not allow private fork, in order to keep commit history of the template repo, we will have to do it by running this script.
**ACTION**: Cloning jos repository (step 3)

Make the script executable with the command

```
chmod +x run.sh
```

Then run it with the command

```
./run.sh
```

If it successfully runs, you will see...

Lastly, switch to lab1 branch by

```
git checkout lab1
```
ACTION: Test your jos

Run the following commands:

```bash
$ cd jos-labs-yourusername
$ make qemu-nox
```

You must see something like following:

```
You may quit qemu by pressing:
```

Ctrl+A, X
**ACTION:** Edit student.info and commit your change

Edit student.info via vim, emacs, nano, etc., e.g.,

```
$ vim student.info (press i to edit and ESC + :wq to write and quit)
```

```
$ nano student.info (stores and quit via pressing Ctrl-X)
```

```
$ emacs student.info
```

Type your information!

```
1 OSU ID (xxx-yyy-zzz) : 933123456
2 ONID ID (e.g., songyip) : songyip
3 Name : Yipeng Song
4 CS 444/544 ? : 444
5 Lab Class # : Lab 1
```
**ACTION**: Commit your change

Run:

```bash
$ git add student.info
$ git commit
  .. type commit message, e.g., edit student.info
$ git push
```
Commit result example

```
1 edit student.info
2 # Please enter the commit message for your changes. Lines starting
3 # with '#' will be ignored, and an empty message aborts the commit.
4 # On branch main
5 # Changes to be committed:
6 # (use "git reset HEAD <file>..." to unstage)
7 #
8 # modified: student.info

os2 ~/cs444/s21/os2-lab1-Rogersyp 172% git push
warning: push.default is unset; its implicit value is changing in
Git 2.0 from 'matching' to 'simple'. To squelch this message
and maintain the current behavior after the default changes, use:

    git config --global push.default matching

To squelch this message and adopt the new behavior now, use:

    git config --global push.default simple

See 'git help config' and search for 'push.default' for further information.
(the 'simple' mode was introduced in Git 1.7.11. Use the similar mode
'current' instead of 'simple' if you sometimes use older versions of Git)

Username for 'https://github.com': Rogersyp
Password for 'https://Rogersyp@github.com':
Counting objects: 5, done.
Delta compression using up to 96 threads.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 298 bytes | 0 bytes/s, done.
Total 3 (delta 2), reused 0 (delta 0)
remote: Resolving deltas: 100% (2/2), completed with 2 local objects.
To https://github.com/OSU-CS444-S21/os2-lab1-Rogersyp.git
  f6e52cf.9ae64fa main --> main
```
How to Start Labs?

Setup lab environment first (we will do this today!)

Read Lab description online

https://classes.engr.oregonstate.edu/eecs/spring2023/cs444-001/labs/Lab1.pdf

Finish all exercises, and run

$ make grade
Running GDB with JOS

Go to jos directory

Use the dual split-screen mode in tmux

Run `make qemu-nox-gdb` on the left side (must run a single instance of qemu..)

   Port bind error could occur if you have another instance of qemu running..

Run `gdb` on the right side (must be under jos directory)

   Otherwise, your gdb will never attach to jos qemu..
Attaching remote gdb to qemu to debug JOS kernel.

Left

```
os2 ~/cs444/s21/os2-lab1-Rogersyp 152% make qemu-nox-gdb
***
*** Now run 'gdb'.
***
quemu-system-i386 -nographic -drive file=obj/kern/kernel.img,index=0,media=disk,format=raw -serial mon:stdio -gdb tcp::26220 -D qemu.log -S
```

Right

```
os2 ~/cs444/s21/os2-lab1-Rogersyp 153% gdb
```
Let's set a breakpoint at the address 0x7c00.

\[ b \ *0x7c00 \]

Then, continue the execution via

\[ c \ (\text{meaning continue..}) \]
You can start Exercise 3 of Lab 1!

Use ‘si’ to follow the function call..
If you are curious about x86 assembly


Search instructions on Google

```
0x00007ccb ? rezn insl (%dx),%es:(%edi)
```

Repeat String Operation Prefix

<table>
<thead>
<tr>
<th>Opcode</th>
<th>Mnemonic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>F3 6C</td>
<td>REPNZ INS m8, DX</td>
<td>Input (E)CX bytes from port DX into ES:[(E)DI].</td>
</tr>
<tr>
<td>F3 6D</td>
<td>REPNZ INS m16, DX</td>
<td>Input (E)CX words from port DX into ES:[(E)DI].</td>
</tr>
<tr>
<td>F3 6D</td>
<td>REPNZ INS m32, DX</td>
<td>Input (E)CX doublewords from port DX into ES:[(E)DI].</td>
</tr>
</tbody>
</table>
Grading Example

Please ignore ‘Clock skew detected’ messages
Example of the correct output of Lab 1

gemu-system-i386 -nographic -drive file=obj/kern/kernel.img,index=0,media=disk,format=raw -serial mon:stdio -gdb tcp::26078 -D qemu.log
444544 decimal is 1544200 octal!
entering test_backtrace 5
entering test_backtrace 4
entering test_backtrace 3
entering test_backtrace 2
entering test_backtrace 1
entering test_backtrace 0
Stack backtrace:
  ebp f010ff18 eip f0100087 args 00000000 00000000 00000000 00000000 f01009db
    kern/init.c:19: test_backtrace+71
  ebp f010ff38 eip f0100069 args 00000000 00000001 f010ff78 00000000 f01009db
    kern/init.c:16: test_backtrace+41
  ebp f010ff58 eip f0100069 args 00000001 00000002 f010ff98 00000000 f01009db
    kern/init.c:16: test_backtrace+41
  ebp f010ff78 eip f0100069 args 00000002 00000003 f010ffb8 00000000 f01009db
    kern/init.c:16: test_backtrace+41
  ebp f010ff98 eip f0100069 args 00000003 00000004 00000000 00000000 00000000
    kern/init.c:16: test_backtrace+41
  ebp f010ffb8 eip f0100069 args 00000004 00000005 00000000 0010094 0010094
    kern/init.c:16: test_backtrace+41
  ebp f010ffd8 eip f01000ea args 00000005 0006c880 00000640 00000000 00000000
    kern/init.c:43: i386_init+77
  ebp f010fff8 eip f010003e args 00111021 00000000 00000000 00000000 00000000
    kern/entry.S:83: <unknown>+0
leaving test_backtrace 0
leaving test_backtrace 1
leaving test_backtrace 2
leaving test_backtrace 3
leaving test_backtrace 4
leaving test_backtrace 5
Welcome to the JOS kernel monitor!
Type 'help' for a list of commands.
K>
How to Submit Labs?

All lab submissions must be turn in via your lab repository on CS444/544 gitlab.

$ git add … (add files to git repo)

$ git commit (commit your changes)

After finishing the lab:

$ git tag lab1-final

$ git push

$ git push origin --tags

This completes the lab. In the jos directory, commit your changes with git commit, git tag lab1-final, git push, and git push origin --tags to submit your code. Please do not forget to create and include the file .lab1-extra in case if you finished extra-credit challenge.

This will push lab1-final to the repository...
How to get help from TA?

- Get on the course Discord server
- Post your question on the each lab channel (JOS Lab1 ~ 4)
- Check TA availability, and then send a DM to a TA
  - Please do not bug our TAs much during their out-of-hour for the TA job. They could help you, but that’s all from their voluntary service. Please send many thanks to our TAs!
- How to code together with a TA?
  - Use the command TA-HELP
Sharing a tmux session with your TA (virtual finger-to-finger meeting with TA)

Copy the following string to TA: /tmp/os2-vIujsog @ os2

Press enter to continue...

TA can share your tmux session and you two can code together
JOS Lab Setup

Tools:

QEMU (Intel 32-bit x86 emulator)

GIT (Source Code Version Control System)

GDB (Debugger)

BASH, TMUX, VIM, etc.

We will use GIT to checkout all code and submit your lab progress!
Read more at...

GIT cheat sheet:  [https://www.git-tower.com/blog/git-cheat-sheet](https://www.git-tower.com/blog/git-cheat-sheet)

VIM cheat sheets:  [https://devhints.io/vim](https://devhints.io/vim) and [https://vim.rtorr.com/](https://vim.rtorr.com/)

GDB cheat sheets:  [https://cs.brown.edu/courses/cs033/docs/guides/gdb.pdf](https://cs.brown.edu/courses/cs033/docs/guides/gdb.pdf)

TMUX cheat sheet:

[https://gist.github.com/MohamedAlaa/2961058](https://gist.github.com/MohamedAlaa/2961058) (the prefix is ` in CS444 settings)