

CS 444/544 OS II

Lab Tutorial #1

Lab Setup, Tools, and Debugging
Prof. Sibir Mohan | Spring 2022

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



Lab instruction website:

https://sibin.github.io/teaching/cs444-osu-operating-systems/spring_2022/lab.html

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.

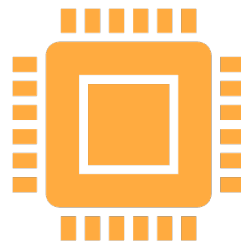
Certainly the definitive reference for x86 assembly language programming is Intel's instruction set architecture reference, which you can find on [the cs444/544 reference page](#) in two flavors: an HTML edition of the old [80386 Programmer's Reference Manual](#), which is much shorter and easier to 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

JOS Lab (lab1-lab4, 70%)



Lab 1: Booting (10%)

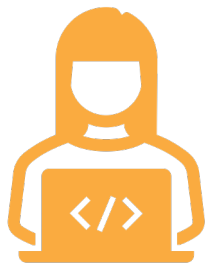
Learn how an OS boots from BIOS to bootloader to the OS kernel



Lab 2: Virtual Memory (15%)

Learn how to manage physical/virtual memory space in an OS kernel

JOS Lab (lab1-lab4, 70%)



Lab 3: User Environment+System Calls (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 date as the lab)

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 beautiful 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.

Solving a challenge adds +1% towards the entire course credit

Today's Tutorial

1. Lab environment setup
2. Commit your information on your own 'jos' repository
3. Run JOS with TMUX

ACTION: Setup lab environment on OS servers

1. Connect to any one of the following servers:

- os2.engr.oregonstate.edu
- oldos2.engr.oregonstate.edu
- oldos1.engr.oregonstate.edu
- os1.engr.oregonstate.edu

2. RUN the following command (**please copy-and-paste**):

```
$ /nfs/farm/classes/eecs/spring2019/cs444-001/cs444-setup.py
```

This will setup BASH, VIM, GDB, QEMU and TMUX

Running the Script

Type 'n' if you wish to keep your dotfiles.

NOTE: Only select 'n' IF you know what you're doing. The safe bet is to pick 'y' for all options.

```
[jangye@flip1 ~]$ /nfs/farm/classes/eecs/spring2019/cs444-001/cs444-setup.py
Cloning into '/nfs/stak/users/jangye/.cs444'...
remote: Enumerating objects: 62, done.
remote: Counting objects: 100% (62/62), done.
remote: Compressing objects: 100% (44/44), done.
remote: Total 409 (delta 32), reused 46 (delta 16), pack-reused 347
Receiving objects: 100% (409/409), 9.29 MiB | 16.17 MiB/s, done.
Resolving deltas: 100% (249/249), done.

Do you want to install peda to ~/.gdbinit (y/n) ?
y
Do you want to install cs444 custom tmux configuration (y/n) ?
y
Do you want to install .bashrc (y/n) ?
y
Do you want to install .vimrc and vim plugins (y/n) ?
y
Error detected while processing /nfs/stak/users/jangye/.vimrc:
line 20:
E185: Cannot find color scheme 'angr'
Press ENTER or type command to continue
```

ACTION: Register your account!

- We will use a private gitlab to host your projects
FERPA Act prevents us from storing your course progress outside OSU
- Visit the website and register an account:
<https://gitlab.unexploitable.systems>
Use **@oregonstate.edu e-mail address** (you can't register otherwise)
- Wait for confirmation e-mail
After confirming the message, **please log on to the gitlab website**

ACTION: Register Your Public Key to Gitlab (Step 1)

- You can **reuse** the public key **if you already have ssh keys on our server**
- Otherwise, please create one by typing the following command:

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

- Print your public key, and then copy the key to the clipboard

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

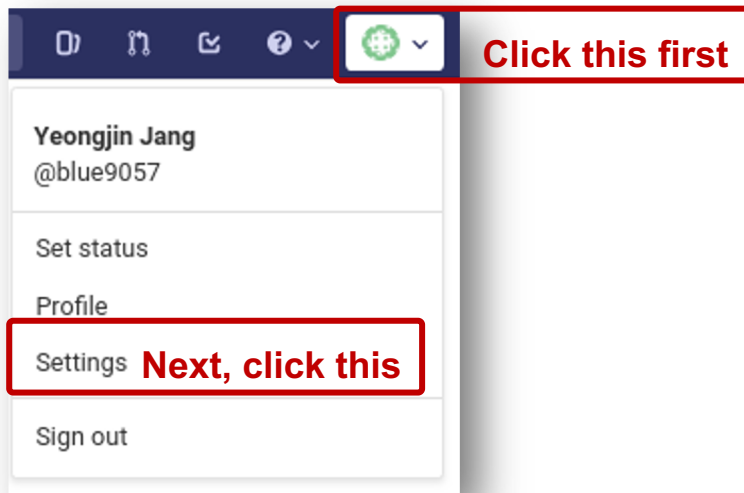
```
ecdsa-sha2-nistp256 (THIS IS A SAMPLE PUBLIC KEY)
```

```
AAAAE2VjZHNhLXNoYTItbmlzdHAyNTYAAAAIbmlzdHAyNTYAAABBBFRxlq/fIouV7Kf1GVEwL04/yIprKdtf9KYO
```

```
Hk8gAbtIxocFFsAgBuEzRg4EtjQEYnitroSm2F14mHy2cz27+ho= jangye@os2.engr.oregonstate.edu
```

Register Your Public Key to Gitlab (Step 2)

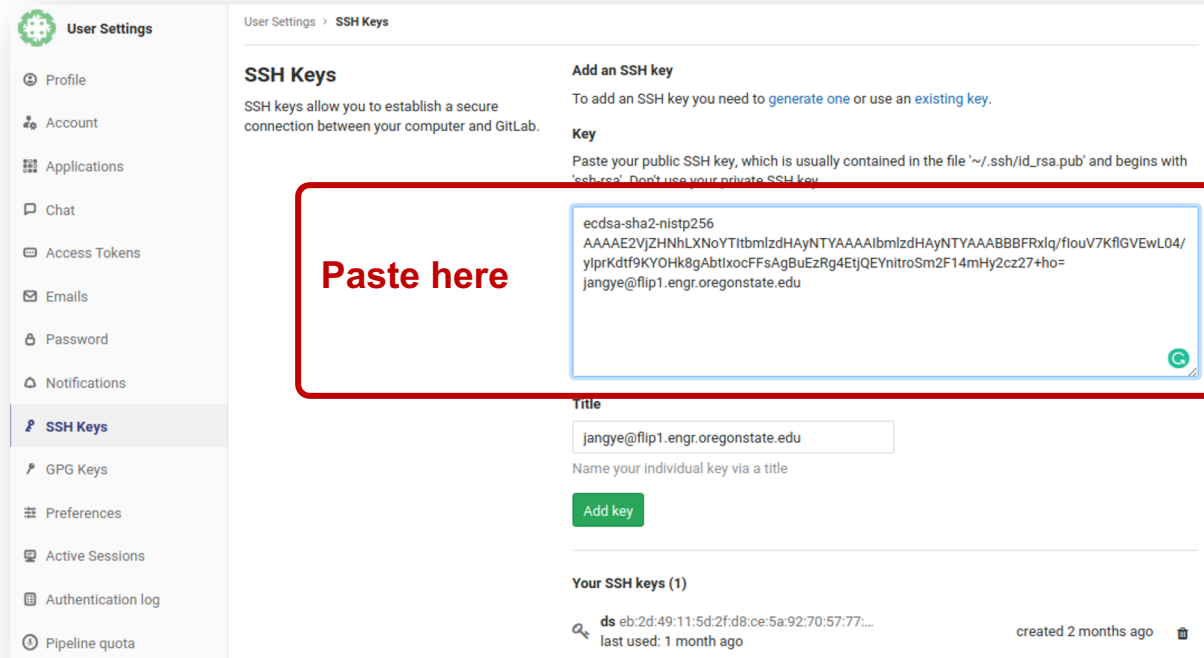
Go to settings page



Register Your Public Key to Gitlab (Step 3)

Go to SSH Key Settings

Paste your key



The screenshot shows the GitLab User Settings page for SSH Keys. On the left is a sidebar with navigation options: Profile, Account, Applications, Chat, Access Tokens, Emails, Password, Notifications, SSH Keys (highlighted), GPG Keys, Preferences, Active Sessions, Authentication log, and Pipeline quota. The main content area is titled 'SSH Keys' and includes instructions on how to add a new key. A red box highlights a text input field containing a public SSH key: `ecdsa-sha2-nistp256 AAAAE2VjZHNhLXNoYTItbmlzdHAyNTYAAAIbmlzdHAyNTYAAABBFRxlq/fiouV7KfIGewL04/yIprKdtf9KYOHk8gAbtlxocFFsAgBuEzRg4EtjQEYnitroSm2F14mHy2cz27+ho=jangye@flip1.engr.oregonstate.edu`. Below the input field is a 'Title' field with the value 'jangye@flip1.engr.oregonstate.edu' and an 'Add key' button. At the bottom, a list of existing SSH keys is shown, including one with a fingerprint of 'eb:2d:49:11:5d:2f:d8:ce:5a:92:70:57:77...' and a creation date of '2 months ago'.

User Settings > SSH Keys

SSH Keys

SSH keys allow you to establish a secure connection between your computer and GitLab.

Add an SSH key

To add an SSH key you need to [generate one](#) or use an [existing key](#).

Key

Paste your public SSH key, which is usually contained in the file `~/ssh/id_rsa.pub` and begins with `'ssh-rsa'`. Don't use your private SSH key.

Paste here

```
ecdsa-sha2-nistp256
AAAAE2VjZHNhLXNoYTItbmlzdHAyNTYAAAIbmlzdHAyNTYAAABBFRxlq/fiouV7KfIGewL04/
yIprKdtf9KYOHk8gAbtlxocFFsAgBuEzRg4EtjQEYnitroSm2F14mHy2cz27+ho=
jangye@flip1.engr.oregonstate.edu
```

Title

Name your individual key via a title

[Add key](#)

Your SSH keys (1)

<code>ds eb:2d:49:11:5d:2f:d8:ce:5a:92:70:57:77...</code>	created 2 months ago
last used: 1 month ago	

Forking the JOS repository

- A **fork** is **making your own copy** of an existing repository
You will work on your JOS repository
- Go to <https://gitlab.unexploitable.systems/root/jos> and fork the repository!

Fork project

A fork is a copy of a project. Forking a repository allows you to make changes without affecting the original project.

Select a namespace to fork the project



Yeongjin Jang

Click this



jos

Project ID: 1



☆ Star 0

🍴 Fork 18

Clone

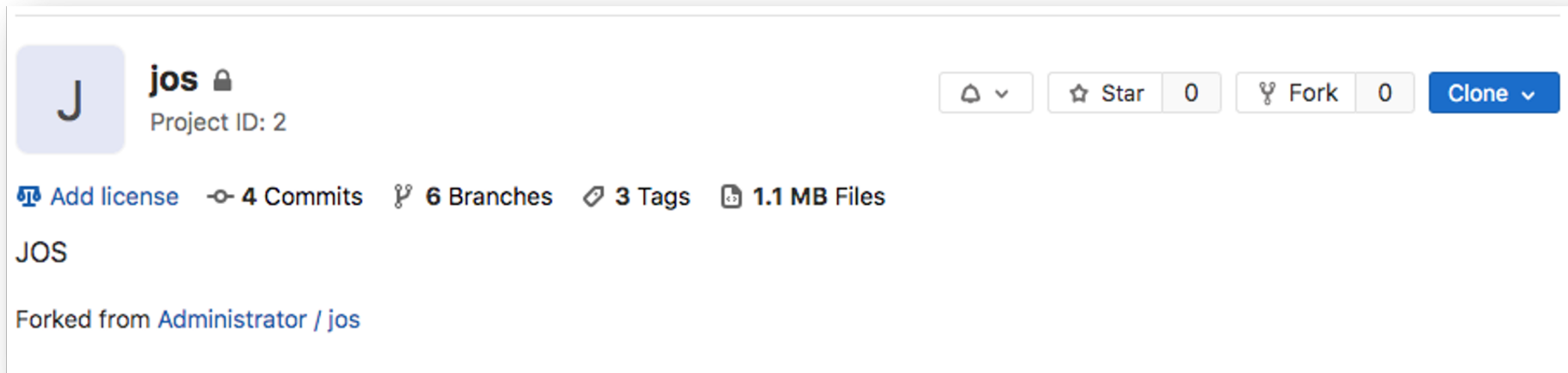
No license. All rights reserved 9 Commits 6 Branches 0 Tags 1.3 MB Files

JOS

Click here

Setting your repository 'Private'

Now you have your own 'jos' repository



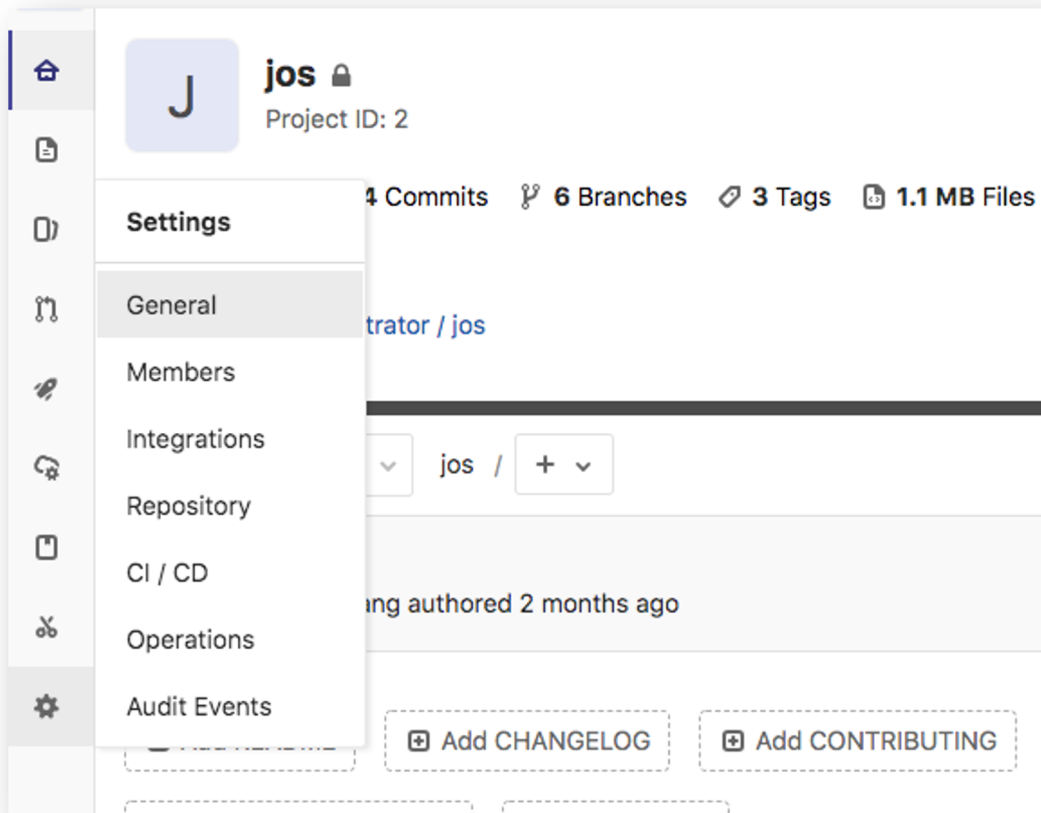
The screenshot shows the GitHub interface for a repository named 'jos'. The repository is currently set to 'Public', as indicated by an open padlock icon. The repository name 'jos' is displayed in bold, followed by a lock icon and the text 'Project ID: 2'. To the right of the repository name are buttons for 'Star' (0), 'Fork' (0), and a blue 'Clone' button with a dropdown arrow. Below the repository name, there are links for 'Add license', '4 Commits', '6 Branches', '3 Tags', and '1.1 MB Files'. The repository name 'JOS' is displayed in a larger font. At the bottom, it says 'Forked from Administrator / jos'.

But right now, your repository is visible to everyone

So, let's make it '**private**' to you

ACTION: Setting your repo private (step 1)

Go to **Settings** → **General**



ACTION: Setting your repo private (step 2)

Expand Visibility settings

Visibility, project features, permissions

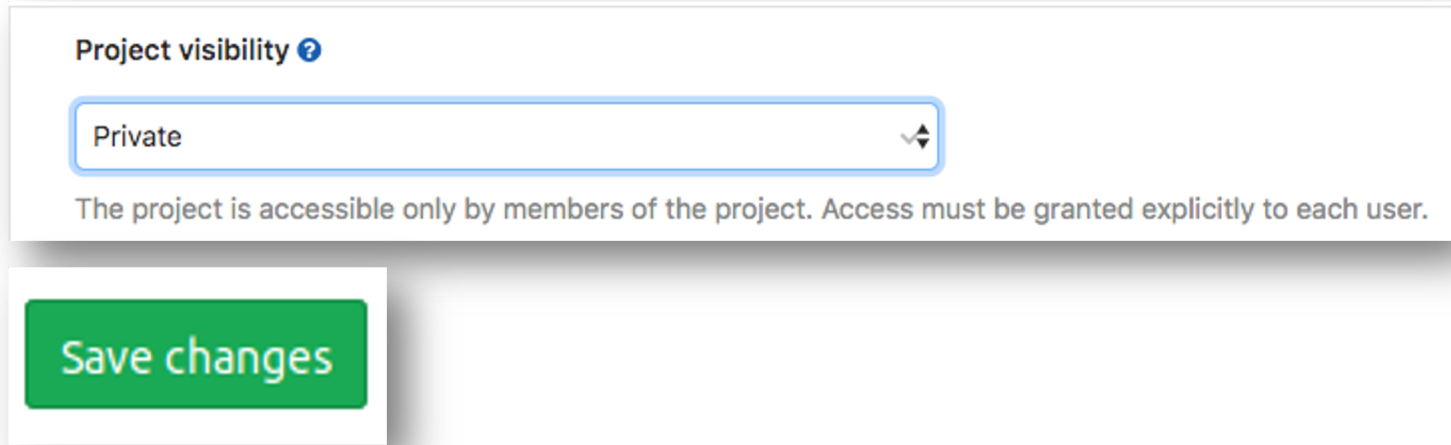
Choose visibility level, enable/disable project features (issues, repository, wiki, snippets) and set permissions.

Expand

Click this

ACTION: Setting your repo private (step 3)

Set your project **'private'** and click **'Save Changes'**



The screenshot shows a form for setting project visibility. At the top, it says "Project visibility" with a help icon. Below that is a dropdown menu with "Private" selected. Underneath the dropdown is a descriptive text: "The project is accessible only by members of the project. Access must be granted explicitly to each user." At the bottom of the form is a green button labeled "Save changes".

Project visibility ?

Private

The project is accessible only by members of the project. Access must be granted explicitly to each user.

Save changes

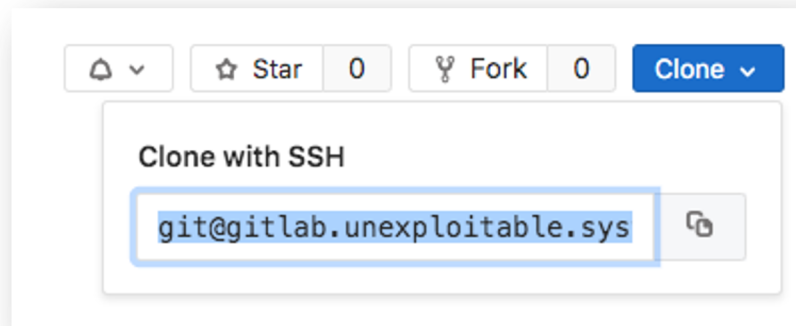
Now only **you, the TAs** and **I** can see your repository.

ACTION: Cloning jos repository (step 1)

Now you need to get the code to the flip/OS server where you can edit it

Click the blue **'Clone'** button and then

Copy the string at the **'Clone with SSH'**



ACTION: Cloning jos repository (step 2)

- Clone the repository by running the following command on flip:

```
$ git clone git@gitlab.unexploitable.systems:your-username-must-be-here/jos.git
```

- **CAUTION:** you need to copy and paste your repository link
e.g., **not root** but **your CS444/544 gitlab ID** must shown on the command line

ACTION: Cloning jos repository (step 3)

- Make the jos directory private to you by running the following command:

```
$ chmod 700 jos
```

- This will make jos private

```
$ ls -l | grep jos
```

```
drwx-----. 1 jangye upg3275 932 Apr  2 02:40 jos
```


NOT READABLE BY OTHERS!

ACTION: Test your jos

- Run the following commands:

```
$ cd jos
```

```
$ make qemu-nox
```

- You must see something like following: 
- You may quit qemu by pressing: **Ctrl+A, X**

```
[jangye@Flip1 (Lab1) ~/jos$] make qemu-nox
make: Warning: File `obj/.deps' has modification time 111 s in the f
+ as kern/entry.S
+ cc kern/entrypgdir.c
+ cc kern/init.c
+ cc kern/console.c
+ cc kern/monitor.c
+ cc kern/printf.c
+ cc kern/kdebug.c
+ cc lib/printfmt.c
+ cc lib/readline.c
+ cc lib/string.c
+ ld obj/kern/kernel
ld: warning: section `.bss' type changed to PROGBITS
+ as boot/boot.S
+ cc -Os boot/main.c
+ ld boot/boot
boot block is 380 bytes (max 510)
+ mk obj/kern/kernel.img
sed "s/localhost:1234/localhost:28275/" < .gdbinit.tmpl > .gdbinit
***
*** Use Ctrl-a x to exit qemu
***
qemu-system-i386 -nographic -drive file=obj/kern/kernel.img,index=0,
444544 decimal is XXX 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
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> █
```

ACTION: Edit student.info and commit your change

- Edit student.info using an editor such as: 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) : 933456789
2 FLIP ID (e.g., jangye) : jangye
3 Name : Yeongjin Jang
4 CS 444/544 ? : 444
5 Lab Class # : Lab 1
6
```

ACTION: Commit your change

- Run the following commands:

```
$ 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 lab1
5 # Changes to be committed:
6 #   (use "git reset HEAD <file>..." to unstage)
7 #
8 #       modified:   student.info
9 #
```

```
[jangye@flip1 (lab1) ~/test/jos$] git commit
[lab1 09da383] edit student.info
 1 file changed, 1 insertion(+), 1 deletion(-)
[jangye@flip1 (lab1) ~/test/jos$] git push
Enter passphrase for key '/nfs/stak/users/jangye/.ssh/id_rsa':
Counting objects: 5, done.
Delta compression using up to 24 threads.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 302 bytes | 0 bytes/s, done.
Total 3 (delta 2), reused 0 (delta 0)
To git@gitlab.unexploitable.systems:red9057/jos.git
af7ec7b..09da383 lab1 -> lab1
```

How to Start Labs?

- Setup lab environment first (we will do this today!)
- Read Lab description online
https://sibin.github.io/teaching/cs444-osu-operating-systems/spring_2022/lab/lab1.html
- Finish all exercises and run
`$ make grade`

Running GDB with JOS

- Go to the '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)
Note: 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

ACTION – TMUX (Step 1)

- We will use 'tmux' to see
 - the **result** of the program 'pointers' and
 - the source code **pointers.c**
- Run the following command:

```
$ tmux
```
- Then press ``` (**backtick**, the key with `~` and on the left side of 1) and `%` (shift-5)
In short, `` + %`

ACTION – TMUX (Step 2)

- Now you see a split screen
- You can move the cursor back and forth by typing
 - `` + ←` (left arrow)
 - `` + →` (right arrow)



Attaching remote gdb to qemu to debug JOS kernel..

Left:

```
[jangye@flip1 (lab1) ~/jos$] make qemu-nox-gdb
***
*** Now run 'make gdb'.
***
qemu-system-i386 -nographic -drive file=obj/kern/kernel.img,index=0,media=disk,format=raw -serial mon:stdio -gdb tcp::28275 -D qemu.log -S
```

Right:

```
[jangye@flip1 (lab1) ~/jos$] gdb
```

Result

Let's set a **breakpoint** at the address **0x7c00**

```
>>> b *0x7c00
```

Then, **continue** execution via

```
>>> c
```

```
[jangye@flip1 (lab1) ~/jos$] gdb
+ target remote localhost:28275
warning: A handler for the OS ABI "GNU/Linux" is not built into this configuration
of GDB. Attempting to continue with the default i8086 settings.

The target architecture is assumed to be i8086
[f000:fff0] 0xffff0: ljmp $0xf000,$0xe05b
0x0000fff0 in ?? ()

Registers
eax 0x00000000    ecx 0x00000000    edx 0x00000663    ebx 0x00000000
esp 0x00000000    ebp 0x00000000    esi 0x00000000    edi 0x00000000
eip 0x0000fff0    eflags [ ]       cs 0x0000f000     ss 0x00000000
ds 0x00000000     es 0x00000000     fs 0x00000000     gs 0x00000000

Assembly
0x0000fff0 ? add    %al,(%bx,%si)
0x0000fff2 ? add    %al,(%bx,%si)
0x0000fff4 ? add    %al,(%bx,%si)
0x0000fff6 ? add    %al,(%bx,%si)
0x0000fff8 ? add    %al,(%bx,%si)
0x0000fffa ? add    %al,(%bx,%si)
0x0000fffc ? add    %al,(%bx,%si)

Source
Stack
[0] from 0x0000fff0
(no arguments)

Memory
Expressions

+ symbol-file obj/kern/kernel
>>> b *0x7c00
Breakpoint 1 at 0x7c00
>>> c
```

You can start **Exercise 3** of Lab 1!

Use 'si' to **follow** the function call

```
Output/messages
[ 0:7c00] => 0x7c00: cli

Breakpoint 1, 0x00007c00 in ?? C)
Registers
eax 0x0000aa55    ecx 0x00000000    edx 0x00000080    ebx 0x00000000
esp 0x00006f20    ebp 0x00000000    esi 0x00000000    edi 0x00000000
eip 0x00007c00    eflags [ IF ]    cs 0x00000000     ss 0x00000000
ds 0x00000000    es 0x00000000    fs 0x00000000    gs 0x00000000

Assembly
0x00007c00 ? cli
0x00007c01 ? cld
0x00007c02 ? xor    %ax,%ax
0x00007c04 ? mov    %ax,%ds
0x00007c06 ? mov    %ax,%es
0x00007c08 ? mov    %ax,%ss
0x00007c0a ? in    $0x64,%al

Source
Stack
[0] from 0x00007c00
(no arguments)
Memory
Expressions
>>> |
```

If you are curious about x86 assembly

- X86 Assembly Guide: <http://flint.cs.yale.edu/cs421/papers/x86-asm/asm.html>
- Search for instructions on Google

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

Repeat String Operation Prefix

Opcode	Mnemonic	Description
F3 6C	REP INS m8, DX	Input (E)CX bytes from port DX into ES:[(E)DI].
F3 6D	REP INS m16, DX	Input (E)CX words from port DX into ES:[(E)DI].
F3 6D	REP INS m32, DX	Input (E)CX doublewords from port DX into ES:[(E)DI].

repnz x86

All Videos News Shopping Images More Settings Tools

About 16,400 results (0.39 seconds)

[REP/REPE/REPZ/REPNE/REPZ: Repeat String Operation Prefix ...](https://c9x.me/x86/html/file_module_x86_id_279.html)
https://c9x.me/x86/html/file_module_x86_id_279.html

x86 assembly tutorials, x86 opcode reference, programming, pastebin with syntax ... and STOS instructions, and the REPE, REPNE, REPZ, and REPZ prefixes ...

Grading Example

```
$ make grade
```

```
running JOS: make[1]: Warning: File `obj/.deps' has modification time 110 s in the future
make[1]: Warning: File `obj/.deps' has modification time 111 s in the future
make[1]: warning: Clock skew detected. Your build may be incomplete.
(0.9s)
  printf: OK
  backtrace count: OK
  backtrace arguments: OK
  backtrace symbols: OK
  backtrace lines: OK
Score: 50/50
make: warning: Clock skew detected. Your build may be incomplete.
```

Please ignore ‘Clock skew detected’ messages

Example of the correct output of Lab 1

```
qemu-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 00010094 00010094
    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 turned in via 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 will push lab1-final to the repository...

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.

How to get help from TAs?

- Get on the course **Discord server** or go to the **office hours**
- Post your question on the specific lab channels (Lab1 ~ 4)
- Check TA availability, and then send a DM to a TA

Please do not bug our TAs much when it is not their office hours. They could help you, but that's their voluntary service.

Please send many thanks to our TAs!

- How to code together with a TA?
 - Use the command **TA-HELP**

ta-help

- Sharing a tmux session with your TA (virtual finger-to-finger meeting with TA)

```
[coe_jangye@os2 ~]$ ta-help  
Copy the following string to TA: /tmp/os2-UvuIUPP @ os2  
Press enter to continue...█
```

- Copy the `tmp` string, and send that to your TA via Discord Direct Message
- Press ENTER (you will see a regular tmux session).
- **TA can share your tmux session and the two of you can code together!**

JOS Lab Setup Summary

- 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!
- Register yourself at: <https://gitlab.unexploitable.systems>

More Resources

- GIT cheat sheet: <https://www.git-tower.com/blog/git-cheat-sheet>
 - VIM cheat sheets: <https://devhints.io/vim> and <https://vim.rtorr.com/>
 - GDB cheat sheets:
<https://cs.brown.edu/courses/cs033/docs/guides/gdb.pdf>
<https://darkdust.net/files/GDB%20Cheat%20Sheet.pdf>
 - TMUX cheat sheet:
<https://gist.github.com/MohamedAlaa/2961058>
- NOTE:** the prefix is ` in CS444/544 settings