

# CS3210: Operating Systems

## Lecture 1

Instructors: Dr. Tim Andersen and Mr. Kyle Harrigan

# What is this course about?

- Continues from CS 2200 Operating Systems
- Application and Lab focused. You learn by doing in this class.
- Forces you to understand how OS's actually work in the real world (no convenient abstractions)
- UNIX/Linux focus
- May be some students' first exposure to the full x86 architecture (bootstrapping, MMU, real mode, paging, traps, etc.)

What is this  
course  
about?

- Gain a detailed knowledge of how an Operating System it built up
- Gain hands-on experience working with a real kernel
- Come away with valuable experience that can be applied to other kernels

Course  
Goals

# Who should be taking this course?

- Useful for those who want to understand systems and platforms
  - E.g., want to improve Android or contribute to Linux kernel
- Those interested in embedded systems
- Those interested low-level programming
- Those who want to write drivers and hardware abstraction layers
- Not good for students who want to work in Java, Python, or other high level software unless you are curious about what's under the hood

# Prerequisites

- C programming (strict)
- CS 2200: Systems and Networks (strict)
- CS 2110: Computer Organization and Programming (recommended)
- CS 3220: Processor Design (recommended)

"I'm doing a (free) operating system (just a hobby, won't be big and professional like gnu) for 386(486) AT clones." -- Linus Torvalds

# What is an operating system?

e.g. OSX, Windows, Linux, FreeBSD, etc.

- What does an OS do for you?
  - **Abstract** the hardware for convenience and portability
  - **Multiplex** the hardware among multiple applications
  - **Isolate** applications to contain bugs
  - Allow **sharing** among applications

# What is an operating system?

## View: layered organization

Layers:

- **User:** applications (e.g., vi and gcc)
- **Kernel:** file system, process, etc.
- **Hardware:** CPU, mem, disk, etc.

-> **Interface** between layers

What is an  
operating  
system?

View:  
layered  
organization

View: core  
services

Typical Core Services:

- Processes
  - Memory
  - File contents
  - Directories and file names
  - Security
  - Many others: users, IPC, network, time, terminals, etc.
- > **Abstraction** for applications



# Example: system calls

- **Interface:** applications talk to an OS via system calls
- **Abstraction:** process and file descriptor

```
fd = open("out", 1);  
write(fd, "hello\n", 6);  
pid = fork();
```

# Why is designing an OS interesting?

- Conflicting design goals and trade-offs
  - Efficient yet portable
  - Powerful yet simple
  - Isolated yet interactable
  - General yet performant
- Some open (ongoing) challenges
  - Security
  - Multi-core

# General information

- Web: <http://cs3210.cc.gatech.edu>
- Piazza: <https://piazza.com/configure-classes/fall2017/cs3210agr>
- GitHub <https://github.gatech.edu/cs3210-fall2017>
- Text: freely available online
  - xv6: a simple, Unix-like teaching operating system
  - (optional) Linux Kernel Development

# General information

- Two instructors for this course:
  - Dr. Tim Andersen
  - Mr. Kyle Harrigan
  - Office hours: TBD
- Two TAs:
  - Meenal Maheshwari
    - Office hours: TBD
  - Anita Ramasamy
    - Office hours: TBD

# Grading policy

- Preparation and In-Class Assignments (10%)
- 2 Quizzes (10% each = 20%)
- Lab (10% each = 50% + 10% bonus)
- Final project (20%)
  - Proposal presentation (5%)
  - Demo & presentation (10%)
  - Write-up (5%)

# Class structure

- Lecture and Demos
- Tutorial
  - Individual exercises
  - Group meeting
- Both meet in Klaus 1456, Tues/Thurs 12:00pm - 1:15pm

Bring your laptop!

# Class structure

<https://cs3210.cc.gatech.edu/cal.html>

- First week:
  - Lecture: about PC, Booting, and C
  - Tutorial: tools
- NOTE: preparation questions and reading

# About labs

- A toy operating system, called JOS (exokernel)
  - Lab 1: Booting a PC
  - Lab 2: Memory management
  - Lab 3: User environments
  - Lab 4: Preemptive multitasking
  - Lab 5: File system and shell
  - Lab 6: Network driver



# About labs

- "Lab 1: Booting a PC" is out (**DUE: Sep 8**)
- Ask questions via Piazza (preferred), Email

## Lab 1: Booting a PC

- Handed out: Tuesday, January 12, 2016
- Due: Tuesday, January 19, 2016

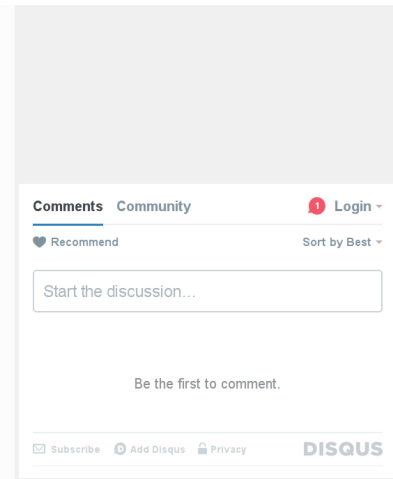
### Introduction

This lab is split into three parts. The first part concentrates on getting familiarized with x86 assembly language, the QEMU x86 emulator, and the PC's power-on bootstrap procedure. The second part examines the boot loader for our cs3210 kernel, which resides in the `boot/` directory of the `lab/` tree. Finally, the third part delves into the initial template for our cs3210 kernel itself, named JOS, which resides in the `kernel/` directory.

### Software Setup

The files you will need for this and subsequent lab assignments in this course are distributed using the [Git](#) version control system. To learn more about Git, take a look at the [Git user's manual](#), or, if you are already familiar with other version control systems, you may find this [CS-oriented overview of Git](#) useful.

The URL for the course Git repository is `git://tc.gtisc.gatech.edu/cs3210-lab`



The screenshot shows a Disqus comment interface. At the top, there are tabs for "Comments" and "Community", and a "Login" button with a red notification icon. Below the tabs, there is a "Recommend" button with a heart icon and a "Sort by Best" dropdown menu. A large text input field contains the placeholder text "Start the discussion...". Below the input field, it says "Be the first to comment." At the bottom, there are links for "Subscribe", "Add Disqus", and "Privacy", followed by the "DISQUS" logo.

# Lab Policies

- Labs build on each other. Important to have labs working completely to do later labs.
- Each lab has challenge questions (5%) and regular questions (2%). That need to be written and turned in with code.
- Labs are committed, tagged, and pushed to a GitHub repo for grading. Please don't email labs. If it's not committed, it will not be graded.
- We are using a Vagrant Trusty32-based configuration on VirtualBox as our official configuration.
  - Pull this file from the GitHub
  - Use another VM at your own risk.
- Each student has assigned GATech GitHub repos for turning in labs and other assignments. Only these will be used for grading.
- Best to commit and push daily -- also best way to share code with the instructors and TAs is GitHub.

# About quiz, project

- Two "quizzes" (in-class, about lec/tut/lab)
  - 80 minutes
- Final project
  - Team project on a topic of your choosing (with approval of instructors)
  - No more than 4 students per team
  - On scale of Lab 6 (Bonus project only this semester)
  - <https://cs3210.cc.gatech.edu/proj.html>
  - Pre-proposal, Team proposal, Demo day, Write-up

# About preparation questions and in-class assignments

- Every lecture and tutorial (**DUE: by 11:59 AM on class day**)
- In-class assignments are due a week from the class day at midnight.
- No late prep questions or in-class submissions accepted.
- Must be turned in on the prep GitHub (<https://github.gatech.edu/cs3210-fall2017/cs3210-prep-YOUR-USERNAME.git>) by the due date/time.

# Class policy

- Late days
  - **Five** days of grace period can be used on one lab deadline (except bonus lab)
  - Grace period is for non-official emergencies (job interviews, broken laptops, etc.)
  - Labs incur 10% per day late penalty otherwise
  - Repeat submissions are allowed (can incur late penalties)
- No cheating
  - Cheating vs. collaboration
  - Write the name(s) of your sources

See, <https://cs3210.cc.gatech.edu/info.html>

# Equipment

- A laptop or other computer is required for this course.
  - Must be able to install the necessary software on it (VirtualBox, QEMU, etc.)
- Let us know in advance if this is a problem.

# Today's agenda

- What is an operating system?
  - Design
  - Goal
  - Role
  - Example: xv6 and JOS
  - Lab Overview

# Challenges in operating systems

- Portability
- Performance
- Reliability
- Security



# Challenges in (practical) operating systems

e.g. Mac OSX, Windows, Linux

- Legacy (compatibility)
- Implementation
- Business

# CS3210: JOS and xv6

- Micro-kernel: JOS (exokernel)
- Monolithic: **xv6** (UNIX-like)
  - Book: **Xv6, a simple Unix-like teaching operating system**
  - Code: **commentary**

Our version (bug fixes as necessary):

```
$ git clone git@github.gatech.edu:cs3210-fall2017/cs3210-xv6-public.git
```

Public version:

```
$ git clone git://github.com/mit-pdos/xv6-public.git
```

# xv6

## What is xv6?

- A re-implementation of Unix Version 6 (1970's era OS)
- Big enough to illustrate concepts, small enough to digest in one semester
- Used as a reference implementation for studying concrete implementations of core concepts
- Why not study Linux?
  - xv6 - 6000 lines
  - Linux v4 (kernel only) - ~165,000 lines!

xv6

JOS

## What is JOS?

- A basic teaching operating system
- Provides a skeleton for implementing the labs, with key pieces left out for you to implement
- Major parts of operating system you will build in JOS
  - Booting
  - Memory management
  - User environments
  - Preemptive multitasking
  - File system, spawn, and shell
  - Network driver
  - Open-ended project
- Resulting operating system will run under QEMU, or any x86-based personal computer

# Lab 1

## Lab 1 - Booting

- Review x86 assembly
- Boot JOS for the first time under QEMU, familiarize with debugger
- Learn / review physical address space during bootup / real mode
- Step through bootup in gdb
- Learn segmentation basics
- Understanding boot loader
- Learn about ELF binary format
- Observe and understand transition from real mode to protected mode
- Become familiar with C calling conventions by implementing your own backtrace

Lab 1

Lab 2

## Lab 2 - Memory Management

- Kernel physical page management
  - Write the physical page allocator (and deallocator...)
- Virtual memory
  - Understand virtual, linear, and physical addresses
  - Implement reference counting, page table management
  - Permissions and fault isolation
  - Initialize kernel address space

Lab 1

Lab 2

Lab 3

## Lab 3 - User Environments

- Basics of getting a "process" running
  - In JOS terminology this is an "environment"
- Allocating, creating, and running environments
- Basic exception and system call handling
  - Interrupt Descriptor Table
  - Task State Segment
  - Trap Frames
  - Page faults, breakpoints, system calls

Lab 1

Lab 2

Lab 3

Lab 4

## Lab 4 - Preemptive Multitasking

- Part A - Add multiprocessor support
  - Round-robin scheduling
  - Environment management
- Part B - Implement fork()
- Part C - Add Inter-Process Communication
  - Also hardware clock interrupts and preemption



Lab 1

Lab 2

Lab 3

Lab 4

Lab 5

## Lab 5 - File system and shell

- Load and run on-disk executables
- Be able to run a shell on the console
- Build a simple read/write file system

Lab 1

Lab 2

Lab 3

Lab 4

Lab 5

Lab 6

## Lab 6 - Network Driver (optional)

- Build a network stack
- Understand QEMU virtual network
- E1000 emulated card
  - PCI Interface
  - Memory-mapped I/O
  - DMA
- Transmitting and receiving packets

# Tools

- The toolchain setup is critical up-front work for this class
  - We will help you in the first tutorial (Thurs)
  - You can develop on whatever you want, and use whatever you want, but our standardized environment and tools will be used for grading. You have been warned (again).

# Tools

## QEMU

### QEMU - Quick EMUlator

- Performs hardware virtualization
- Can emulate obscure processor architectures
- Does share code with some other projects (VirtualBox)
- Good for working with the kernel (GDB stub)
  - We will use it for this

Tools

QEMU

git

## git - the stupid content tracker

- Created by Linus Torvalds in 2005 for development of the Linux kernel
- Of course, widely used now
- All of our configuration management, assignment turn-in, etc. will be done using git.
- Commit early, commit often, push often!
  - We can help you better if we can see your code
  - We can grant you partial credit if you miss deadlines
- It is very easy for us to compare work across students
  - Don't cheat!

Tools

QEMU

git

**gdb**

## **gdb - GNU project debugger**

- Hopefully you have experience...
- You may be exposed to some new features of gdb
  - Kernel / Remote debugging
  - QEMU
  - Esoteric (but useful) commands
- This isn't an IDE (though the TUI is close)

Tools

QEMU

git

gdb

Vagrant

## Vagrant

"Create and configure lightweight, reproducible, and portable development environments"

- A convenient front-end to provide reproducible virtual machines
- Easily hooks into Virtualbox, Parallels, VMWare, etc.
- Our vagrant instance on [github.gatech.edu](https://github.com/gatech) will be the de-facto machine for grading
  - Test with it before and after tagging your submissions!

# Why using C for OS development?

- Portability
- No runtime
- Direct hardware/memory access
- (decent) Usability



# How are your C skills?

- C is the most important skill to have for this class
- OS programming involves a great deal of pointer arithmetic. The compiler will not save you from these mistakes.
- If your C skills are rusty, you may struggle with the lab work.
- Labs deadlines come quickly. Little time to build basic skills.

# Prep quiz: the C programming language

- Open your laptop
- <http://cs3210.cc.gatech.edu/q/prep.txt>
- Download the quiz
- Please submit to GitHub by 1 week from today (Aug 29 before midnight).

# Selected feedback from last semester

"The amount you learn in this class is astronomical compared to many other classes at GT. Also, the drop rate is insane, but please don't water down the class because of this. It was a hard class but it should remain a hard class. I kind of like the if you actually make it through you probably get an A/B. There seem to be a number of classes at tech where you can earn a C in a class but really can't apply any of it (admittedly, been there done that). I like the fact that anyone who makes it through this class will have no significant problems writing C at the system level. So I guess for the better, the difficulty was a good aspect of the class."

# Selected feedback from last semester

"The beginning was rough because they kinda assumed we knew stuff about boot processes and computer architecture and that had me lost for several weeks."

- Goal: Hands-on walkthrough

"Text was pretty good, but certain visual aids would greatly help the learning process if provided. Diagrams of memory layout / structures. Flow charts/maps for various processes (trap/interrupt/syscalls, context switching, etc.). When first starting the course with zero experience with operating systems, it was a little overwhelming at first, and I had to go through the source code to figure things out, which took more time than ideal."

- Goal: MORE PICTURES

"The lectures and pre-lecture questions were essentially useless"

- Goal: More demos, hands-on

Bottom line: Expect less powerpoint! (today excluded)

# Next lecture

- Tutorial: Group, Tools, Lab1
- Register **Piazza**
- **Lab 1: Booting a PC** is out (**DUE: 11:59 PM, Sep 8**)
- Don't forget to submit "preparation question" (**DUE: 11:59 AM, 24 Aug**)

# References

- UW CSE 451
- OSPP
- MIT 6.828
- Wikipedia
- The Internet

