Introduction

Advanced Operating Systems

Overview

- ~~JOS~~ -> OpenLSD
- Setting up the environment
- Source code overview

OpenLSD

- “Borrowed” from MIT 6.828
  - With **heavy** modifications
- Framework with “boring stuff” (later!)
- **Incremental** per lab
  - Patch per lab via git on top of your own changes
- Infrastructure for **testing** and **handing in**

Setting up

Dependencies (Ubuntu)

```
build-essential gdb qemu git
```

Download and compile

```
git clone https://github.com/vusec/aos-labs.git
cd aos-labs && ./setup.sh
source .settings
make
```

Running your kernel

QEMU emulator

```
make qemu
```

- Output via serial (terminal) and VGA (qemu console)
- Interactive monitor for debugging:
  - See `help` command
  - ...or you can add your own

More options...

QEMU without GUI

```
make qemu-nox
```

- No VGA/qemu console, only serial in terminal
- Exit with `ctrl-a x`

More options...

QEMU with GDB

`make qemu-gdb`

- **Attach** with `make gdb`
- All usual gdb commands work (`break`, `continue`, `examine`, `backtrace`, ...)
- More during the GDB lecture!
- (also without GUI via `make qemu-nox-gdb`)

Triple faults...

- During first labs, your kernel does not set up any **exception handling**
- Thus, errors cause your VM to **triple fault** (i.e., **reset**)
- QEMU is configured to print “`Triple fault`” and dump CPU state (registers) before reset
- Patched QEMU that halts execution before reset: https://sipb.mit.edu/iap/6.828/tools/

OpenLSD overview

`boot/`	Bootloader
`kernel/`	Kernel code
`include/`	Header files public APIs kernel
`lib/`	Shared user/kernel code (e.g., strings)
`user/`	User programs (lab 3+)
`obj/`	Compiler output

OpenLSD overview

>

```
runcmd (...)  at kernel/monitor.c
monitor (...) at kernel/monitor.c
_panic (...)  at kernel/main.c
mem_init ()   at kernel/mem/init.c
kmain ()      at kernel/main.c
<unknown> ()  at kernel/boot.S
```

Other files of interest

`include/types.h`	Basic types and macros
`include/x86-64/paging.h`	Descriptor and page table definitions
`include/x86-64/memory.h`	Description of segments, virtual memory layout and page info

Handing in labs

```
make grade
```

Run tests.

```
make tarball
```

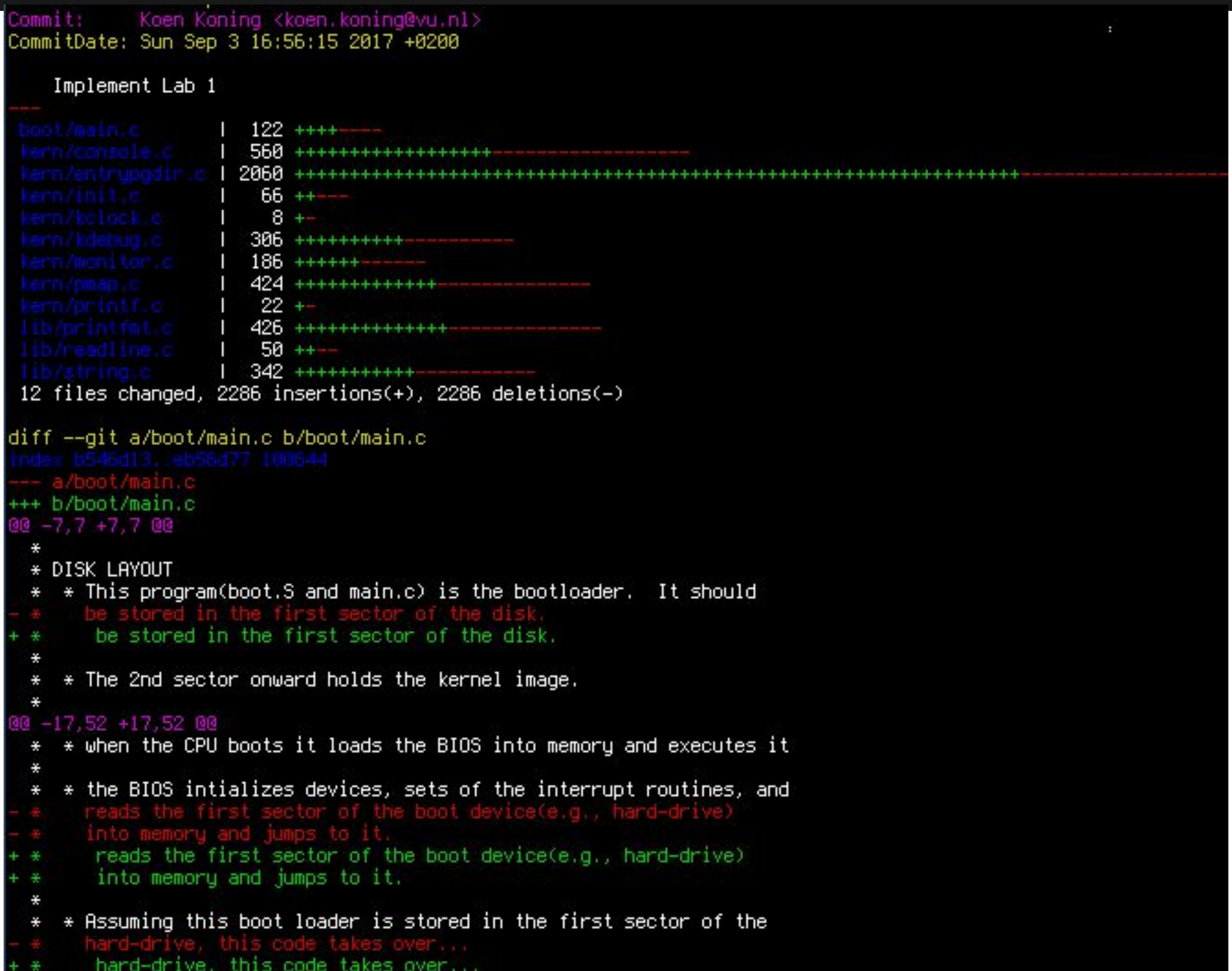
Creates archive containing your entire git repo.

Please keep git history sane :-)

Submit **only** this on Canvas

(preferably 1 submission per team)

Don’t do this...



```
Commit:     Koen Koning <koen.koning@vu.nl>
CommitDate: Sun Sep 3 16:56:15 2017 +0200

    Implement Lab 1
---
 boot/main.c         |  122 ++++----
 kern/console.c      |  560 ++++++++++++++++++------------------------
 kern/entrypgdir.c   | 2060 +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++-------------------
 kern/init.c         |   66 ++---
 kern/kclock.c       |    8 +-
 kern/kdebug.c       |  306 +++++++++++----------
 kern/monitor.c      |  186 ++++++------
 kern/pmap.c         |  424 +++++++++++++--------------------
 kern/printf.c       |   22 +-
 lib/printfmt.c      |  426 ++++++++++++++--------------
 lib/readline.c      |   50 ++--
 lib/string.c        |  342 +++++++++++------------
 12 files changed, 2286 insertions(+), 2286 deletions(-)

diff --git a/boot/main.c b/boot/main.c
index b546d13..eb56d77 100644
--- a/boot/main.c
+++ b/boot/main.c
@@ -7,7 +7,7 @@
  *
  * DISK LAYOUT
  *  * This program(boot.S and main.c) is the bootloader.  It should
- *    be stored in the first sector of the disk.
+ *     be stored in the first sector of the disk.
  *
  *  * The 2nd sector onward holds the kernel image.
  *
@@ -17,52 +17,52 @@
  *  * when the CPU boots it loads the BIOS into memory and executes it
  *
  *  * the BIOS intializes devices, sets of the interrupt routines, and
- *    reads the first sector of the boot device(e.g., hard-drive)
- *    into memory and jumps to it.
+ *     reads the first sector of the boot device(e.g., hard-drive)
+ *     into memory and jumps to it.
  *
  *  * Assuming this boot loader is stored in the first sector of the
- *    hard-drive, this code takes over...
+ *     hard-drive, this code takes over...
```

Changing our code

Don’t break our commands (`make grade` etc.)

Split off coding convention changes into separate commits.

Better: stick to our coding convention

(will save you a lot of trouble down the line!)

Pulling in new labs

```
git add -p
git commit -m “Finished lab1”
git fetch --all origin
(... Make a backup ...)
git rebase origin/lab2
```

Support

- Use the **discussion board** on Canvas!
- Allows us to efficiently answer questions

Rules

- Assignments handed in in groups of 2
- Canvas -> People -> Group
- Sign up ASAP, use discussion board to find a teammate