



UCF

**College of Engineering
and Computer Science**

UNIVERSITY OF CENTRAL FLORIDA

Intro to OS and Loaders

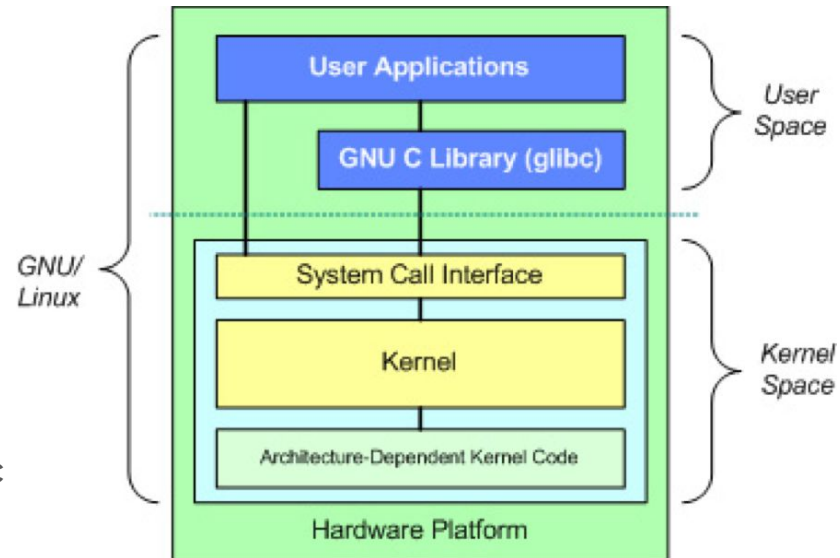
COP-3402 Systems Software
Paul Gazzillo



UCF

Computing Infrastructure Uses a Layered Design

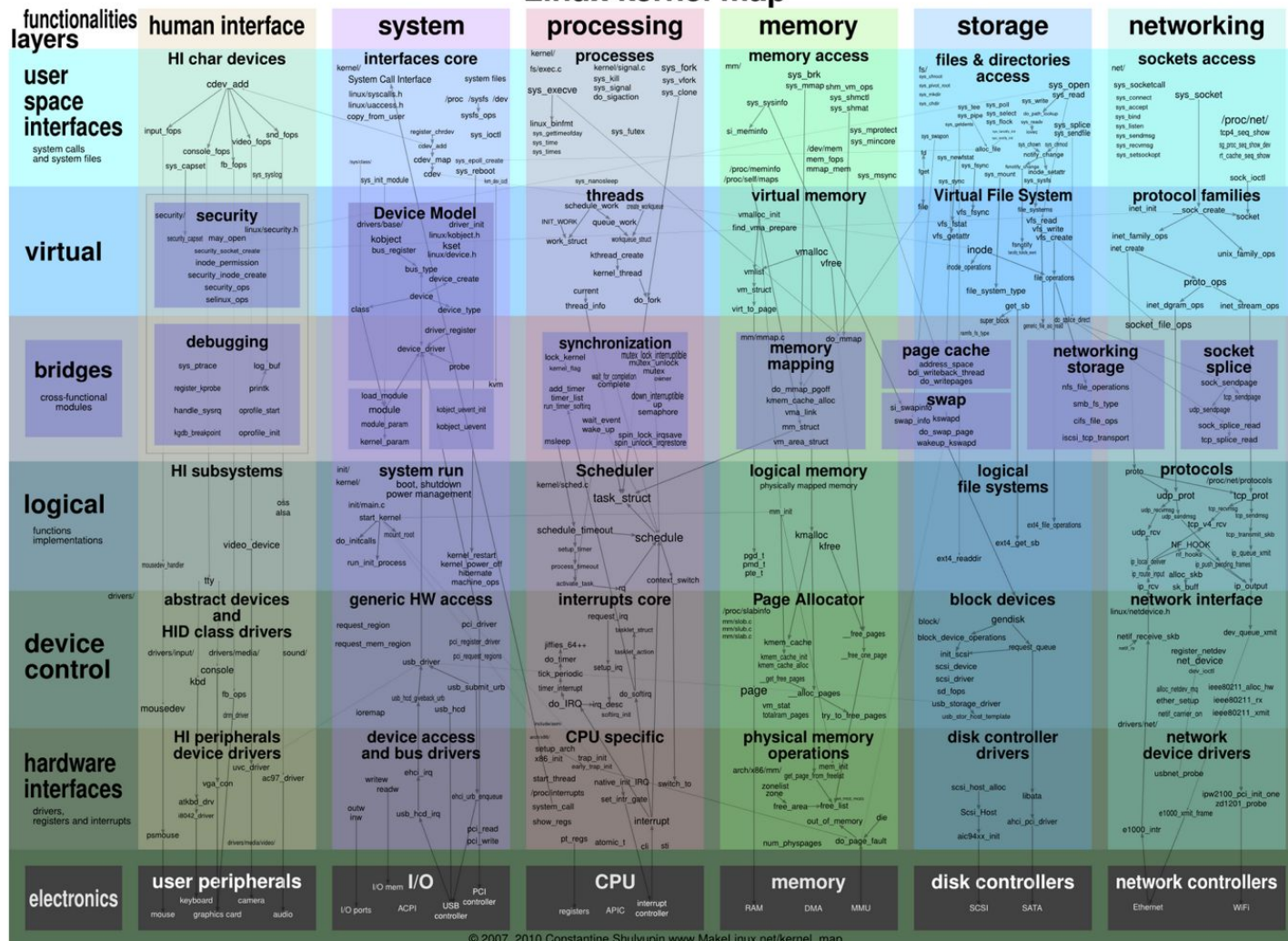
- Hardware
- Firmware (BIOS, UEFI, etc)
 - Programs on ROM, boots an OS
- OS Kernel
 - Abstracts away hardware differences
 - Manages RAM and processor access
- Systems software
 - Libraries, shell, compiler, linker, loader, etc
- Applications



The OS Kernel

- Abstracts away hardware differences
 - Analogy: electric vs internal combustion cars
 - Underlying technology is different
 - User interface is the *same*: steer wheel, pedals
 - Example: persistent storage
 - USB flash drives, SATA solid state drives, EIDE hard disks
 - Same interface to programmer/user: open, read, write, etc
- Manages access to resources
 - Virtual memory: applications request access to use more RAM
 - Processor time: kernel schedules programs to share processor

Linux kernel map

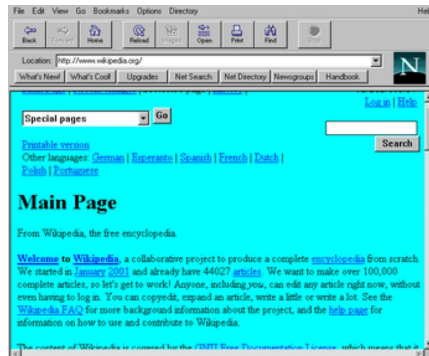


What Is an Operating System

- Possible definitions
 - Just the kernel
 - Kernel and systems software
 - Kernel, systems software, and windowing interface
 - Kernel, systems software, windowing interface, applications
- Where is the line between OS & applications?

Browser Wars: Netscape vs Microsoft 1995-01

- Netscape had a large market share
- Microsoft bundled Internet Explore (IE)
- Opposing claims
 - Microsoft is using monopoly power to undermine competition
 - IE is an integral to the OS as memory management
- Bill Gates' argument
 - <https://m.youtube.com/watch?v=8Lbfcyh8dCM&t=6m30s>

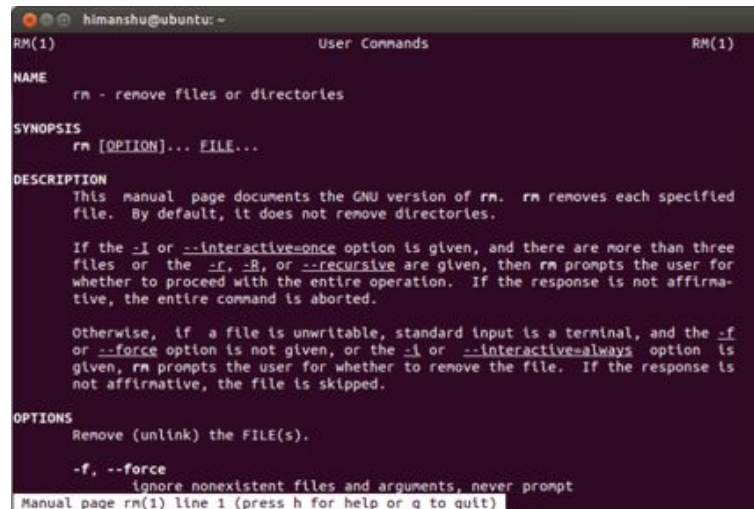


6



Kernel vs. the Operating System

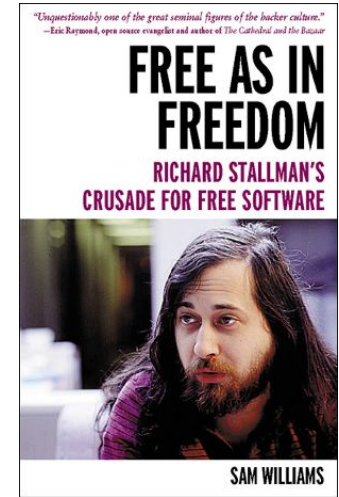
- The command-line is not Linux per se
- The Linux codebase contains only the *kernel*
 - No command-line tools (bash, ls, cd, etc)
 - No windowing interface (MS Windows has this in the kernel)
 - No compiler, linker, loader, libraries, etc



```
himanshu@ubuntu: ~  
RM(1)                                User Commands                                RM(1)  
  
NAME  
    rm - remove files or directories  
  
SYNOPSIS  
    rm [OPTION]... FILE...  
  
DESCRIPTION  
    This manual page documents the GNU version of rm.  rm removes each specified  
    file.  By default, it does not remove directories.  
  
    If the -I or --interactive=once option is given, and there are more than three  
    files or the -r, -R, or --recursive are given, then rm prompts the user for  
    whether to proceed with the entire operation.  If the response is not affirma-  
    tive, the entire command is aborted.  
  
    Otherwise, if a file is unwritable, standard input is a terminal, and the -f  
    or --force option is not given, or the -i or --interactive=always option is  
    given, rm prompts the user for whether to remove the file.  If the response is  
    not affirmative, the file is skipped.  
  
OPTIONS  
    Remove (unlink) the FILE(s).  
  
    -f, --force  
        ignore nonexistent files and arguments, never prompt  
Manual page rm(1) line 1 (press h for help or q to quit)
```

GNU = GNU's Not Unix

- 1983 Stallman announces GNU
 - Free, complete, Unix-like operating system
 - Unix is proprietary, owned by AT&T
- 1984: development begins
 - gcc, glibc, GNOME, bash, binutils, coreutils, etc
- 1985: the GNU Manifesto, Free Software Foundation
 - FLOSS - free/libre open-source software
- 1989: GNU Public License
 - Free as in speech (libre), not beer (gratis)
- 1990: GNU Hurd kernel
 - Most of OS done, except kernel and drivers



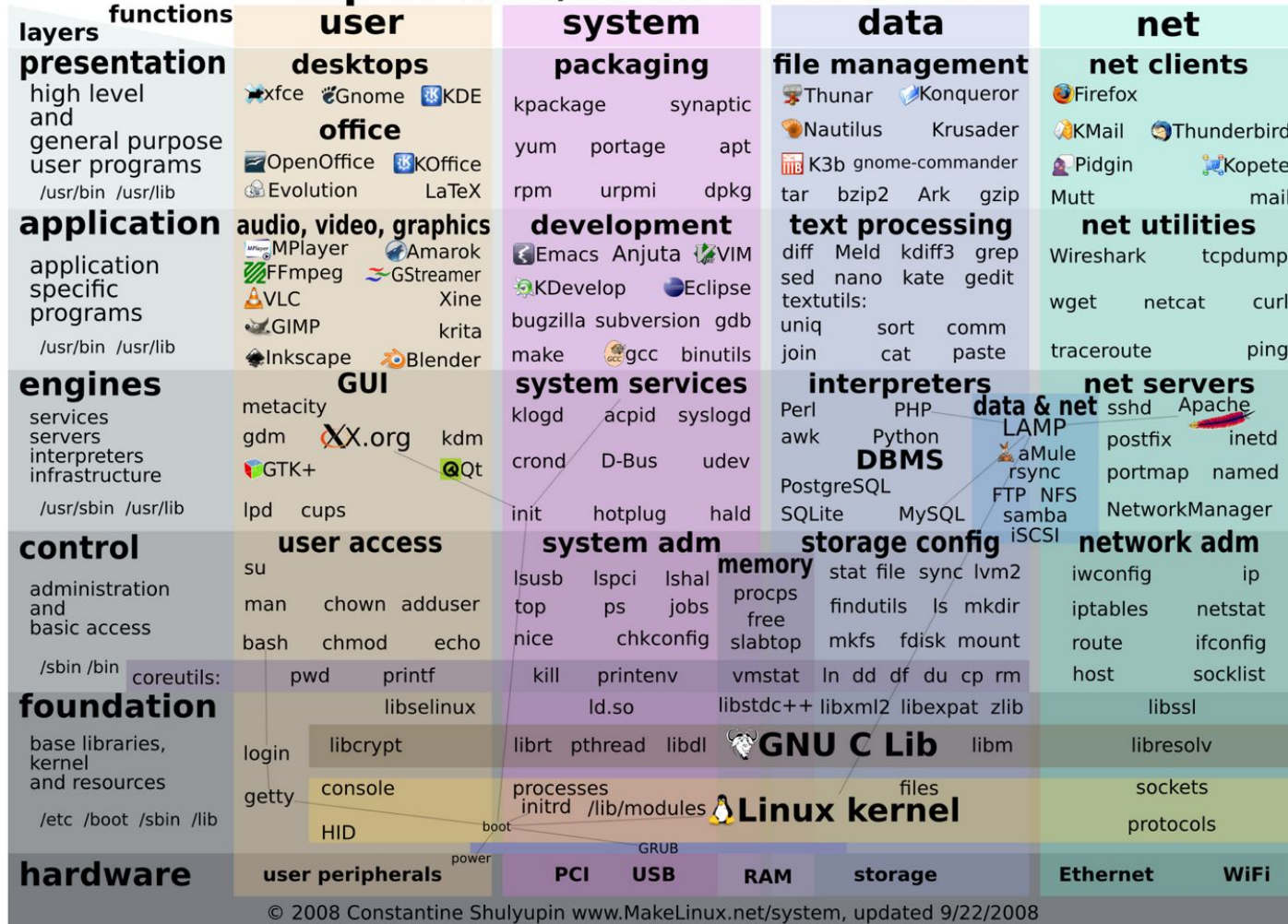
Linux Fills in the Missing Piece

- 1991: Linux Torvalds announces project, 50 kLoC
- 1992: licenses it under GNU Public License (GPL)
- 1992: “Linux is obsolete” - Prof. Tannebaum
 - microkernel vs. monolithic kernel debate
- 1993 and on: distributions of complete OS
 - Slackware, Debian, Red Hat, etc
- 2003: SCO sues distributors
 - claimed rights over Unix trademark and source code
- 2007: Android announced, uses Linux kernel
- 2011: Linux 3.0
- 2019: Linux is in 100s of millions of devices; over 10mil LoC



https://en.wikipedia.org/wiki/Linus_Torvalds

Map of GNU/Linux OS and FOSS



<http://www.makelinux.net/system/>

GNU/Linux Naming Controversy

- Linux filled a gap in free software
 - GNU Hurd is still rarely used
- Most Linux-based OSes use GNU system software
 - Except Android, some routers, and others
- “GNU/Linux”

“Most of the tools used with linux are GNU software and are under the GNU copyleft” – Torvalds, 1992

“Today tens of millions of users are using an operating system that was developed so they could have freedom—but they don't know this, because they think the system is Linux and that it was developed by a student 'just for fun'.” – Stallman, 2012

- Just “Linux”

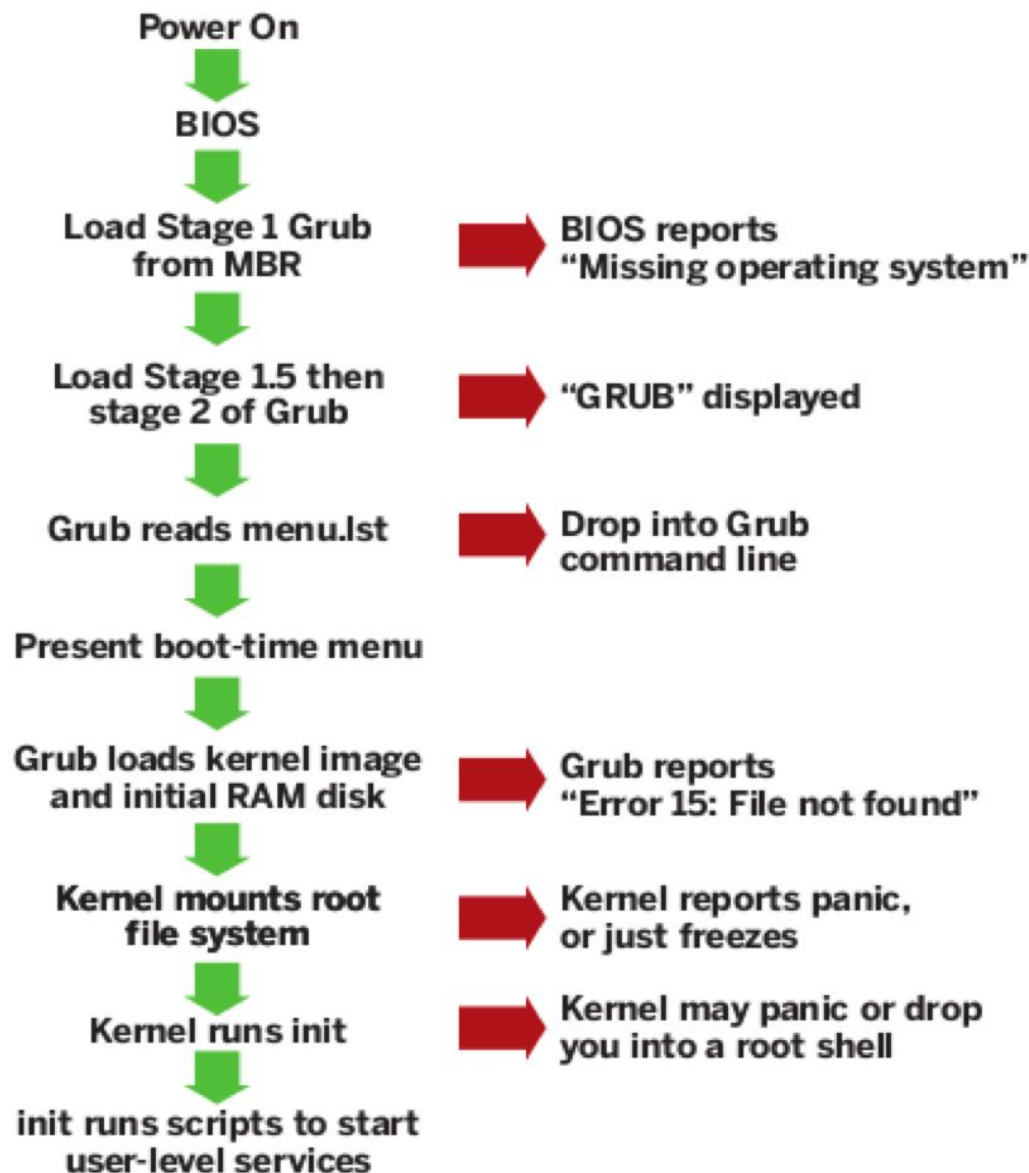
“This claim is a proxy for an underlying territorial dispute; people who insist on the term GNU/Linux want the FSF to get most of the credit for Linux because [Stallman] and friends wrote many of its user-level tools.” – Raymond

“Well, I think it's justified, but it's justified if you actually make a GNU distribution of Linux ... because if you actually make your own distribution of Linux, you get to name the thing, but calling Linux in general 'GNU/Linux' I think is just ridiculous” – Torvalds, 2001

From Hardware to Software: How Your Program Gets Executed

Booting: Starting the First Program

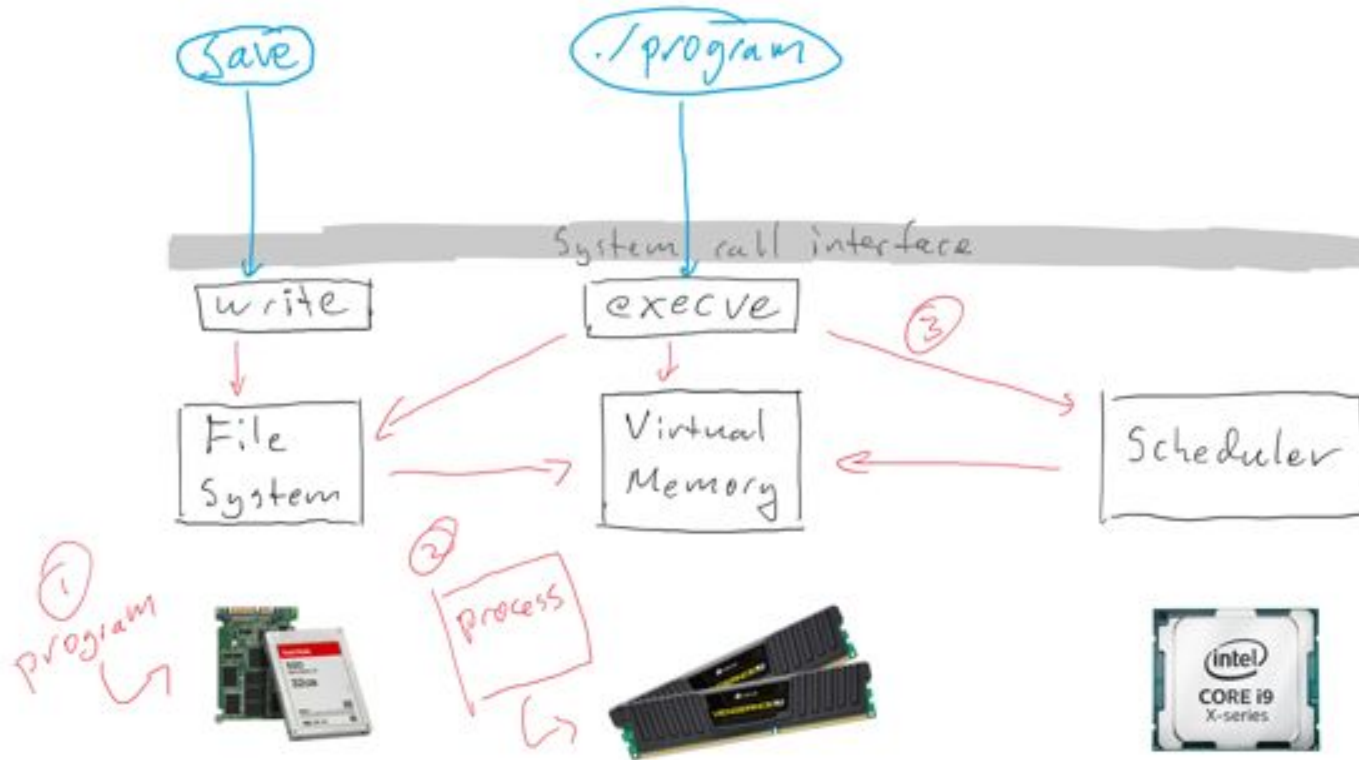
- Booting is a series of steps that launches the OS
 - “Pull yourself up by your own bootstraps”
- Firmware stored on motherboard always runs first
 - BIOS in the original PC and for several decades
 - UEFI more recently
- BIOS then loads and runs first 512 bytes of hard disk
- UEFI has device drivers, runs bootloader program
- Bootloader initializes and runs OS kernel
- Kernel runs its first process



The Loader: Asking the OS to Run a Program

- Compiler preparation
 - Object file layout (ELF in Linux)
 - C runtime library setups up environment and calls main()
- exec family of Linux system calls
 - Read program binary from storage
 - Organize binary in RAM (ELF layout)
 - Branch to _start (C runtime library defines start)
- C runtime sets up environment and calls main()
- More details: <http://tldp.org/LDP/LG/issue84/hawk.html>

The Loader: Asking the OS to Run a Program



Demo: objdump

- Disassemble (-d)
- Symbol table (-t)
- Linking (undef functions)
 - Two C files, one defines and one calls the function

Build Automation

Build Automation Is Essential

- Everyone has used gcc
 - `gcc -o program file.c`
- Tedious to run by hand every time
- Large programs use separate compilation
 - Multiple C files: `gcc -c file.c` (creates `file.o`)
 - Multiple calls to compiler/linker: `gcc -o program file1.o file2.o`
- Build automation: repeatable, programmable build
 - `make` has been around for decades and is widely used
 - It's *incremental*: rebuilds only as needed
 - **make is mandatory for your project**

syllabus/projects/make/Makefile

Variable

```
SRC = $(wildcard *.c)
OBJ = $(SRC:%.c=%.o)
PROG = simplec
```

Gets all .c files
and .o files

```
.PHONY: all clean
```

Build rule
target: source
commands

```
all: $(PROG)
```

Creates final
program

```
$(PROG): $(OBJ)
        $(CC) -o $@ $^
```

Must indent
with tab

```
%.o: %.c
        $(CC) -c $<
```

Compiles any .c
files to .o files

```
clean:
        rm -rf $(PROG) $(OBJ)
```

Always include a
clean target

Demo: Using `make`

- Initial build, everything compiled and linked
- Modify a source file
 - Only that file is recompiled and linked
- `touch` an object file
 - Only linking happens

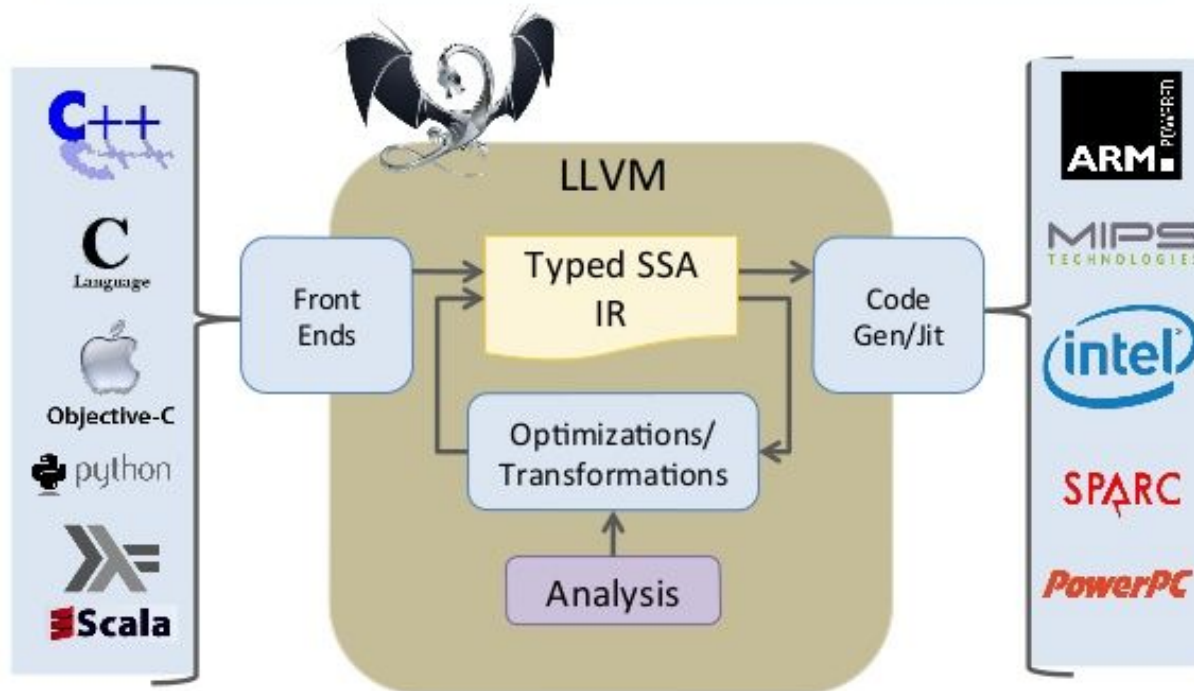
Running LLVM IR

LLVM = Low-Level Virtual Machine

- Started by Chris Lattner in during his masters studies
 - Open-source
 - Hundreds of contributors
 - Lattner now at Apple
- Defines a new intermediate representation (IR)
- IR is like a high-level assembly language
 - Has function definitions, types, and unlimited registers
- Makes writing new compilers easier
 - Target LLVM and reuse existing machine code generation

Separates Compiler Front-End and Back-End

- Front-end source processing
- Back-end machine code generation



Example Program: helloworld.ll

- Function definitions
- Function calls
- Variables (SSA form, only assign variables once)
- Assembly-like operations

```
@.str = private unnamed_addr constant [15 x i8] c"hello, world!\0A\00", align 1

define i32 @main() #0 {
    %1 = alloca i32, align 4
    store i32 0, i32* %1, align 4
    %2 = call i32 @printf(i8* getelementptr inbounds ([15 x i8],
                                                    [15 x i8]* @.str, i32 0, i32 0))
    ret i32 0
}

declare i32 @printf(i8*, ...) #1
```

Generating Machine Code from LLVM IR

- LLVM IR “compiles” to machine code
- Can use clang, LLVM’s C front-end (handles linking)
 - `clang -o helloworld helloworld.ll`
- Run as usual: `./helloworld`
- Your compiler will output LLVM IR
 - clang/LLVM will generate the machine code

Demo: Compiling LLVM IR

Conclusion

- Basic OS and software infrastructure
- Linking and loading
- Makefiles
- Compiling and running LLVM IR
- Next time: compiler project overview and project 0