WHY DO YOU HAVE TO TAKE THIS STUPID CLASS

• People don’t just write programs in one language for one platform anymore. Real projects have lots of parts.
WHY DO YOU HAVE TO TAKE THIS STUPID CLASS
WHY DO YOU HAVE TO TAKE THIS STUPID CLASS

• People don’t just write programs in one language for one platform anymore. Real projects have lots of parts.
• Computers are changing: parallelism is much more important today than it was in the 90s.
• Stuff you learn here will be used in security, OS, etc.
WHAT IS THIS GUY DOING?

UNIVAC, 1951
<table>
<thead>
<tr>
<th>Process Name</th>
<th>% CPU</th>
<th>CPU Time</th>
<th>Threads</th>
<th>Idles Wake Ups</th>
</tr>
</thead>
<tbody>
<tr>
<td>vmware-vmx</td>
<td>38.0</td>
<td>1:26:25.83</td>
<td>37</td>
<td>2014</td>
</tr>
<tr>
<td>Activity Monitor</td>
<td>13.7</td>
<td>1:45:25.29</td>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>WindowServer</td>
<td>5.3</td>
<td>4:34:13.48</td>
<td>10</td>
<td>13</td>
</tr>
<tr>
<td>launchservicesd</td>
<td>2.1</td>
<td>28:51.68</td>
<td>8</td>
<td>0</td>
</tr>
<tr>
<td>kernel_task</td>
<td>1.5</td>
<td>3:49:40.17</td>
<td>352</td>
<td>410</td>
</tr>
<tr>
<td>sysmond</td>
<td>0.7</td>
<td>2:16:17.10</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>Creative Cloud</td>
<td>0.6</td>
<td>25:51.18</td>
<td>24</td>
<td>0</td>
</tr>
<tr>
<td><a href="https://www.amazon.com">https://www.amazon.com</a></td>
<td>0.5</td>
<td>57.63</td>
<td>17</td>
<td>1</td>
</tr>
<tr>
<td>foeserted</td>
<td>0.4</td>
<td>21:08:09.10</td>
<td>10</td>
<td>3</td>
</tr>
<tr>
<td>hidc</td>
<td>0.4</td>
<td>18:07:03</td>
<td>7</td>
<td>0</td>
</tr>
<tr>
<td>VMware Fusion Applications Menu Helper</td>
<td>0.3</td>
<td>1:25:11.45</td>
<td>19</td>
<td>42</td>
</tr>
<tr>
<td>launchd</td>
<td>0.3</td>
<td>59:31.68</td>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>Finder</td>
<td>0.2</td>
<td>2:10:66</td>
<td>9</td>
<td>2</td>
</tr>
<tr>
<td>tcod</td>
<td>0.2</td>
<td>36:35</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>VMware Fusion Applications Menu</td>
<td>0.2</td>
<td>58:12.71</td>
<td>28</td>
<td>34</td>
</tr>
<tr>
<td>loginwindow</td>
<td>0.2</td>
<td>2:05:73</td>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>VMware Fusion</td>
<td>0.2</td>
<td>1:35:24.75</td>
<td>15</td>
<td>5</td>
</tr>
<tr>
<td>VMware Fusion Applications Menu Helper</td>
<td>0.2</td>
<td>44:29.44</td>
<td>5</td>
<td>19</td>
</tr>
<tr>
<td>coreaudiod</td>
<td>0.2</td>
<td>17:34:39</td>
<td>40</td>
<td>40</td>
</tr>
<tr>
<td>QuickLookSatellite</td>
<td>0.2</td>
<td>16.63</td>
<td>11</td>
<td>1</td>
</tr>
<tr>
<td>AppleUserECM</td>
<td>0.2</td>
<td>19.92</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>logd</td>
<td>0.1</td>
<td>3:26:21.25</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>fsest</td>
<td>0.1</td>
<td>23:10.54</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>

System: 5.99%
User: 3.05%
Idle: 90.96%

CPU Load

Threads: 2,530
Processes: 590
What is an Operating System?

• **Referee**
  – Manage sharing of resources, Protection, Isolation
    » Resource allocation, isolation, communication

• **Illusionist**
  – Provide clean, easy to use abstractions of physical resources
    » Infinite memory, dedicated machine
    » Higher level objects: files, users, messages
    » Masking limitations, virtualization

• **Glue**
  – Common services
    » Storage, Window system, Networking
    » Sharing, Authorization
    » Look and feel
OS PROVIDES ABSTRACTIONS THAT ARE BETTER THAN THE UNDERLYING HARDWARE TO REDUCE COMPLEXITY

- Processor → Thread
- Memory → Address Space
- Disks/SSD → Files
- Networks → Sockets
- Machines → Processes
Across incredibly diversity

Computers Per Person

1:10^6

1:10^3

1:1

10^3:1

years

Bell's Law: new computer class per 10 years

Number crunching, Data Storage, Massive Inet Services, ML, …

Productivity, Interactive

Streaming from/to the physical world

The Internet of Things!

Mote!
Moore’s Law – The number of transistors on integrated circuit chips (1971-2018)

Moore’s law describes the empirical regularity that the number of transistors on integrated circuits doubles approximately every two years. This advancement is important as other aspects of technological progress – such as processing speed or the price of electronic products – are linked to Moore's law.

Moore’s Law Officially Ended in 2016: No longer doubling transistor density every 18-24 months
# Vast Range of Timescales

**Jeff Dean’s “Numbers Everyone Should Know”**

<table>
<thead>
<tr>
<th>Operation</th>
<th>Time (ns)</th>
</tr>
</thead>
<tbody>
<tr>
<td>L1 cache reference</td>
<td>0.5</td>
</tr>
<tr>
<td>Branch mispredict</td>
<td>5</td>
</tr>
<tr>
<td>L2 cache reference</td>
<td>7</td>
</tr>
<tr>
<td>Mutex lock/unlock</td>
<td>25</td>
</tr>
<tr>
<td>Main memory reference</td>
<td>100</td>
</tr>
<tr>
<td>Compress 1K bytes with Zippy</td>
<td>3,000</td>
</tr>
<tr>
<td>Send 2K bytes over 1 Gbps network</td>
<td>20,000</td>
</tr>
<tr>
<td>Read 1 MB sequentially from memory</td>
<td>250,000</td>
</tr>
<tr>
<td>Round trip within same datacenter</td>
<td>500,000</td>
</tr>
<tr>
<td>Disk seek</td>
<td>10,000,000</td>
</tr>
<tr>
<td>Read 1 MB sequentially from disk</td>
<td>20,000,000</td>
</tr>
<tr>
<td>Send packet CA-&gt;Netherlands-&gt;CA</td>
<td>150,000,000</td>
</tr>
</tbody>
</table>
OPERATING SYSTEMS HELP MANAGE COMPLEXITY

• Advances in hardware make programming difficult
  • OS Provides Consistent Abstractions
  • OS Manages Resource Sharing

• Key Building Blocks:
  • Processes
  • Threads, Concurrency, Scheduling, Coordination
  • Address Spaces
  • Protection, Isolation, Security
  • Communication
  • Persistent Storage, transactions, consistency, resilience
  • Interfaces to Devices
In 2011, smartphone shipments exceeded PC shipments!

2011 shipments:
- 487M smartphones
- 414M PC clients
  - 210M notebooks
  - 112M desktops
  - 63M tablets
- 25M smart TVs

4 billion phones in the world → smartphones over next few years

Then… Not Only PCs connected to the Internet!
The world is a large distributed system
- Microprocessors in everything
- Vast infrastructure behind them

Internet Connectivity

MEMS for Sensor Nets

Databases
Information Collection
Remote Storage
Online Games
Commerce

Scalable, Reliable, Secure Services
WHAT IS IN THE OS?

- Components:
  - Memory Management
  - I/O Management
  - CPU Scheduling
  - Communications? (Email?)
  - Multitasking?
- What About:
  - File System?
  - Multimedia Support?
  - User Interface/Windowing?
  - Internet Browser?

- There’s no universally-accepted definition.
- The one program that runs all the time is the kernel.
- Maybe you can say “everything that comes with a fresh OS install”
- *Studying OSes is really about the Hardware/Software interface (API)* - John Kubiatowicz
POLICY/MECHANISM

- **Goal:**
  - Keep user programs from crashing the OS
  - Keep user programs from crashing each other

- **Policy:**
  - Programs are not allowed to read/write memory of their programs or of the OS

- **Mechanism:**
  - Address translation
  - Dual-mode operation
1. Rosetta
2. mac OS Port
1. OS
2. Driver Support
android 13

go edition
Given we have a single processor cache that is

- 32-bit address space
- Word addressed (addresses are left shifted by 2 by adding "00" to end of address inside the processor, this implies that it can address $2^32 \times 4 = 16\text{GBytes of memory}$)
- Cache is 16KByte in size
- Cache block size (aka cache line size) = 16 words (64 bytes = $16 \times 4$)
  - # of cache blocks = 256
- direct mapped (1-way associative)

From the above information, we can infer that the offset requires 4 bits ($2^4 = 16$), the index requires 8 bits ($2^8 = 256$), and tag is 20 bits.

From the testing perspective, what are the interesting cases we would want to test about the operation of the cache? What may be some corner cases?

Write a test program that generates addresses to access the cache while hitting the interesting cases and corner cases of the cache.

What makes a good test?

- Random traffic
- Hits corner cases (interesting scenarios a totally random test will not activate)
- Hits the corner cases randomly rather than explicitly
- Come up with a reasonable number of cycles to test with each type of random traffic to get a good tradeoff between compute resource and test thoroughness.

For the sake of simplicity, we will not put the checker code in this test program (assume the correctness will be checked elsewhere), and for sake of the problem, we will not be testing the data part of the program. In another word, this is a cache traffic driver program. The checker code will be placed elsewhere.

Take as much time as you want, but I'm expecting people to only spend 20-40 minutes on this.
TURNING IN ASSIGNMENTS:

• We will use GitHub Classroom. See course webpage for link.
• Fill out the survey on the course website (see schedule for today).
CODING GUIDELINES:

• Make sure you test code a bit at a time—split into functions.
• Build pieces one at a time.
• Plan first.
HOMEWORK

• Class will be front-loaded with homework
• Each week you will have two assignments

Homework Assignment
Adding a feature to your kernel

“In-Class” Activity
Informal coding practice
PROGRAMMING IN C

• You’re supposed to kinda know how to write C code
• You need to get good at writing C fast
• C refresher available at:
  https://os.neilklingensmith.com
THE TEXTBOOK

- **Free @** http://ostep.org
- Links to relevant chapters on course webpage schedule
### FILE COMMANDS

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ls</code></td>
<td>directory listing</td>
</tr>
<tr>
<td><code>ls -l</code></td>
<td>formatted listing with hidden files</td>
</tr>
<tr>
<td><code>cd</code></td>
<td>change directory to dir</td>
</tr>
<tr>
<td><code>pwd</code></td>
<td>show current directory</td>
</tr>
<tr>
<td><code>mkdir</code></td>
<td>create directory dir</td>
</tr>
<tr>
<td><code>rm</code></td>
<td>delete file</td>
</tr>
<tr>
<td><code>rm -r</code></td>
<td>delete directory dir</td>
</tr>
<tr>
<td><code>rm -f</code></td>
<td>force remove file</td>
</tr>
<tr>
<td><code>rm -rf</code></td>
<td>remove directory dir</td>
</tr>
<tr>
<td><code>cp</code></td>
<td>copy file to file</td>
</tr>
<tr>
<td><code>mv</code></td>
<td>rename file to file</td>
</tr>
<tr>
<td><code>ln -s</code></td>
<td>create symbolic link 'link' to file</td>
</tr>
<tr>
<td><code>touch</code></td>
<td>create or update file</td>
</tr>
<tr>
<td><code>cat</code></td>
<td>place standard output into file</td>
</tr>
<tr>
<td><code>more</code></td>
<td>output the contents of the file</td>
</tr>
<tr>
<td><code>less</code></td>
<td>output the contents of the file</td>
</tr>
<tr>
<td><code>head</code></td>
<td>output first 10 lines of file</td>
</tr>
<tr>
<td><code>tail</code></td>
<td>output last 10 lines of file</td>
</tr>
<tr>
<td><code>tar</code></td>
<td>output contents of file as it grows</td>
</tr>
</tbody>
</table>

### SSH

- ssh user@host - connect to host as user
- ssh -p port user@host - connect using port p

### INSTALLATION

- `./configure`
- `make`
- `make install`

### NETWORK

- ping host - ping host 'host'
- whois domain - get whois for domain
- dig domain - get DNS for domain
- dig -x host - reverse lookup host
- wget file - download file
- wget -c file - continue download
- wget -r url - recursively download files from url

### SYSTEM INFO

- `date` - show current date/time
- `cal` - show this month's calendar
- `uptime` - show uptime
- `w` - display who is online
- `whoami` - who are you logged in as
- `uname -a` - show kernel config
- `cat /proc/cpuinfo` - cpu info
- `cat /proc/meminfo` - memory information
- `man command` - show manual for command
- `df` - show disk usage
- `du` - show directory space usage
- `du -sh` - human readable size in GB
- `free` - show memory and swap usage
- `whereis app` - show possible locations of app
- `which app` - which app will be run by default

### SEARCHING

- `grep pattern files - search for pattern in files`
- `grep -r pattern dir - search recursively for pattern in dir`
- `locate file` - find all instances of file

### PROCESS MANAGEMENT

- `ps -aux - display currently active processes`
- `ps aux - display ps with a lot of detail`
- `kill - send signal to process`
- `killall - kill all processes named proc`
- `bg` - lists background jobs, resumes stopped jobs in the background
- `fg` - bring most recent job to foreground
- `fg n` - bring job n to foreground

### FILE PERMISSIONS

- `chmod octal file - change permission of file`
- `4 - read (r)`
- `2 - write (w)`
- `1 - execute (x)`
- `order: owner/group/world`
- `eg:`
- `chmod 777` - rw for everyone
- `chmod 755` - rw for owner, rx for group/world

### COMPRESS

- `tar cf file.tar files - tar files into file.tar`
- `tar xf file.tar - untar into current directory`
- `tar tf file.tar - show contents of archive`

### TARFLAGS

- `c` - create archive
- `j` - bzip2 compression
- `t` - table of contents
- `k` - do not overwrite
- `x` - extract
- `f` - specify filename
- `w` - ask for confirmation
- `z` - use gzip/gzip
- `v` - verbose

- `gzip file` - compress file and rename to file.gz
- `gzip -d file.gz` - decompress file.gz

### SHORTCUTS

- `ctrl+c` - halt current command
- `ctrl+z` - stops current command
- `fg` - resume stopped command in foreground
- `bg` - resume stopped command in background
- `ctrl+d` - log out of current session
- `ctrl+w` - erase one word in current line
- `ctrl+u` - erase whole line
- `ctrl+r` - reverse lookup of previous commands
- `!!` - repeat last command
- `exit` - log out of current session
No quizzes or exams. Your whole grade is based on homework and final project.

No partial credit for code that doesn’t compile.

Start homework on Tuesday/Wednesday so you can get help on Thursday in lab if you get stuck.

Slop Days: Everyone gets 5 slop days. Each slop day allows you to turn in an assignment 24 hours late.

<table>
<thead>
<tr>
<th>Category</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>Homework</td>
<td>60%</td>
</tr>
<tr>
<td>Participation</td>
<td>20%</td>
</tr>
<tr>
<td>Quizzes</td>
<td>20%</td>
</tr>
</tbody>
</table>
OFFICE HOURS

• Wednesday 1-2:15PM
MICROSOFT TEAMS

• Join Link on Course Website
WHAT ARE WE GOING TO BE DOING…?
ACCESSING THE CLASS SERVER

ssh UVID@cs310.cs.luc.edu
password: 12345678

Note: It’s only accessible from campus.
You also have access to VMware Fusion/Workstation.
THE TEAM

• Microsoft Teams Join Code 34iz1ae
• Join link on course website
GENTOO
WHAT ARE WE GOING TO BE DOING…?

• VMware
BOOTLOADERS
PROGRAMMER’S MODEL OF 386

CPU

Memory

0xFFFFFFFF

0x00000000
PROGRAMMER’S MODEL OF 386

Data Registers
- EAX
- EBX
- ECX
- EDX

Address Registers
- ESI
- EDI
- EBP
- ESP
- EIP

Memory
- 0x00000000
- 0xFFFFFFFF
## PROGRAMMER’S MODEL OF 386: INSIDE THE CPU

<table>
<thead>
<tr>
<th>Data Registers</th>
<th>Address Registers</th>
</tr>
</thead>
<tbody>
<tr>
<td>EAX</td>
<td>ESI</td>
</tr>
<tr>
<td>EBX</td>
<td>EDI</td>
</tr>
<tr>
<td>ECX</td>
<td>EBP</td>
</tr>
<tr>
<td>EDX</td>
<td>ESP</td>
</tr>
<tr>
<td></td>
<td>EIP</td>
</tr>
</tbody>
</table>
PROGRAMMER’S MODEL OF 386: INSIDE THE CPU

Data Registers

EAX

EBX

ECX

EDX

Address Registers

ESI

EDI

EBP

ESP

EIP

main:
mov eax,00000100h
mov ebx,00000200h
loop:
add eax,ebx
cmp eax,00000400h
jlt loop
ret
PROGRAMMER’S MODEL OF 386: INSIDE THE CPU

```
main:
  mov eax,00000100h
  mov ebx,00000200h
loop:
  add eax,ebx
  cmp eax,00000400h
  jlt loop
  ret
```
PROGRAMMER’S MODEL OF 386: INSIDE THE CPU

**Data Registers**
- EAX: 00000100
- EBX: 00000200
- ECX: 
- EDX: 

**Address Registers**
- ESI: 
- EDI: 
- EBP: 
- ESP: 
- EIP: 

```
main:
  mov eax,00000100h
  mov ebx,00000200h
loop:
  add eax,ebx
  cmp eax,00000400h
  jlt loop
ret
```
PROGRAMMER’S MODEL OF 386: INSIDE THE CPU

Data Registers
- EAX: 000000300
- EBX: 000000200
- ECX: 
- EDX: 

Address Registers
- ESI: 
- EDI: 
- EBP: 
- ESP: 
- EIP: 

main:
- mov eax,00000100h
- mov ebx,00000200h
loop:
- add eax, ebx
- cmp eax, 00000400h
- jlt loop
- ret
WHERE SHOULD THE PROGRAM LIVE IN MEMORY?
Say I decide to put my program at \(0x80000000\)
How does it get there?

<table>
<thead>
<tr>
<th>Data Registers</th>
<th>Address Registers</th>
</tr>
</thead>
<tbody>
<tr>
<td>EAX</td>
<td>00000300</td>
</tr>
<tr>
<td>EBX</td>
<td>00000200</td>
</tr>
<tr>
<td>ECX</td>
<td></td>
</tr>
<tr>
<td>EDX</td>
<td></td>
</tr>
</tbody>
</table>

```
main:
0x08000000   mov eax,00000100h
0x08000006   mov ebx,00000200h
loop:
0x08000008   add eax,ebx
0x0800000A   cmp eax,00000400h
0x0800000C   jlt loop
0x0800000E   ret
```
SO HOW DOES THE OS GET INTO MEMORY?

AS WE’LL SEE, OUR HARDWARE CHOICES ARE NOT AWESOME…
**DDR SDRAM (Main Memory)**

- **Volatile**
  - Loses its contents on poweroff
  - Must be re-initialized on each boot
- **Read/Write**

**Flash Memory**

- **Nonvolatile**
  - Retains its contents on poweroff
- **Read Only**
  - Can’t use for variable storage
1. Hardware reads BIOS from flash chip into DRAM.

CPU

Memory

BIOS

Flash chip holds BIOS

0xFFFFFFFF

0x00000000

Hardware reads BIOS from flash chip into DRAM.
2. BIOS reads MBR from disk into DRAM
3. MBR reads the bootloader from disk into memory.
4. Bootloader loads the OS kernel into memory and starts the kernel.
CPU

0x00000000

0xFFFFFFFF

Flash chip holds BIOS

Now Kernel has control!

Memory

Kernel
Bootloader
MBR
BIOS

0x00000000

2TB SSD

SAMSUNG
Disk is divided into 512-byte sectors 256 GByte

Sector = 2,097,152 Sectors

First 2048 sectors (1 Mbyte) store bootloader
WRITING AN MBR
BIOS Is Kinda Like a Set of Drivers for MBR

MBR is only 512 bytes!
THE ONLY THING A COMPUTER KNOWS HOW TO DO IS EXECUTE INSTRUCTIONS.

```
if( a < 5 ) {
    b += a;
    a++;
}
```

```
cmp ax,5
jge .not_less_than
add bx,ax
inc ax

.not_less_than:
...```
KINDS OF INSTRUCTIONS

- Arithmetic
  - Add, subtract, multiply, divide
- Logic
  - AND, OR, NOT, XOR
- Shifts
  - Left shift, right shift, rotate, etc.

- Control
  - Branch/Jump
  - Procedure calls
- Memory Accesses
  - Load/store
THE ONLY THING A COMPUTER KNOWS HOW TO DO IS EXECUTE INSTRUCTIONS.
RASPBERRY PI BOOT PROCESS
<table>
<thead>
<tr>
<th>CPU</th>
<th>GPU</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Memory</td>
<td></td>
</tr>
</tbody>
</table>
CPU

GPU

bootcode.bin  (grub on PC)

Memory

boot partition

swap partition

rootfs
CPU

GPU

bootcode.bin

kernel8.img

Memory

boot partition

swap partition

rootfs
```plaintext
CPU
kernel8.img

GPU

init

Memory

SanDisk 32 GB microSD

boot partition

swap partition

rootfs
```