CMPSC 311 - Introduction to Systems Programming

Module: UNIX/Operating Systems

Professor Patrick McDaniel
Fall 2013
TODO:

1. Get and install license and software from VM (if needed), see email

2. Get most recent version (13.04, NOT 12.04) of Ubuntu from the “Download” page of the website
   - Make sure you get the right bit version of your hardware (most recent hardware is 64-bit, but many laptops still have 32-bits)

3. Install an virtual machine.
UNIX

• Developed in 1969 at Bell Labs
  ‣ originally intended for use as a programmer environment for developing multi-platform code
    • Its use grew quickly and the architectural advantages were embraced by the academic and industrial communities.
    • It dominates the “big iron” industrial environments
      ‣ About 2/3 of servers run some variant of UNIX (2013)

• Main attributes
  ‣ *multiuser* - supports multiple users on the system at the same time, each working with their own terminal
  ‣ *multitasking* - support multiple programs at a time
  ‣ *portability* - when moving from hardware to hardware, only the lowest layers of the software need to be reimplemented.
UNIX

- AT&T research systems, (1969)
  - Ken Thompson, Dennis Ritchie, Brian Kernighan, Doug McIlroy, etc.
- System III, 1981
- System V, 1983; System V Release 4, 1988-95
- SCO UnixWare - the former Microsoft Xenix, 1980 (became) SCO in 1987
- Univ. of California, Berkeley - BSD series, up to 1995
  - FreeBSD, NetBSD, OpenBSD
- Mach (kernel), 1985-94
- Sun Microsystems, SunOS, Solaris, OpenSolaris (became Oracle in 2010)
- IBM, AIX, z/OS (which is a mainframe operating system)
- Silicon Graphics Inc., IRIX
- Hewlett-Packard, HP-UX
- Digital Equipment Corp. / Compaq / HP, Ultrix, Digital Unix, Tru64 Unix
- DEC/IBM/HP consortium, OSF/1 (based on Mach)
- Apple, Mac OS X (derived from FreeBSD and Mach)
Linux

• Linux, since 1991
  ‣ Open source version of UNIX that has seen broad adoption in academic and industrial communities

• GNU/Linux
  ‣ Red Hat, SUSE/Novell, Caldera (defunct, SCO), Debian, Mandrake/Mandriva, Slackware, Gentoo, Ubuntu, Knoppix, Fedora, etc., etc.

• Android, since 2003 - Linux kernel, Open Handset Alliance, Android Open Source Project
Open Source

• Many UNIX systems in use today are distributed as “open source”
  ‣ *Open source software is distributed with a license where the copyright allows the user of the source to review, modify, and distribute with no cost to anyone.*

• Variants of this arrangement allow a person (a) to derive software from the distribution and recharge or (b) never charge anyone for derivative works.

*Aside:* free beer vs free speech (gratis vs. libre)?
UNIX

- UNIX can be viewed as software layers
  - **OS kernel** --- direct interaction with hardware
  - **system calls** --- interface to the kernel
  - **system libraries** --- wrappers around system calls
  - **programming language libraries** --- extends system libraries
  - **system utilities** --- application-independent tools
    - e.g., fsck, fdisk, ifconfig, mknod, mount, nfsd
  - **command interpreter**, command shell --- user interface
  - **application libraries** --- application-specific tools
  - **applications** --- complete programs for ordinary users
    - some applications have their own command shells and programming-language facilities (e.g., Perl, Python)
What’s an OS?

• Software that:

  1. Directly interacts with the hardware
     • OS is trusted to do so; user-level programs are not
     • OS must be ported to new HW; user-level programs are portable
  2. Manages (allocates, schedules, protects) hardware resources
     • decides which programs can access which files, memory locations, pixels on the screen, etc., and when
  3. Abstracts away messy hardware devices
     • provides high-level, convenient, portable abstractions
       ‣ e.g., files vs. disk blocks

UNIX is a classical example of an OS.
UNIX is an abstraction provider

- The OS is the “layer below”
  - a module that your program can call (with system calls)
  - provides a powerful API (the UNIX OS API)

---

A process running your program

<table>
<thead>
<tr>
<th>OS API</th>
<th>OS</th>
</tr>
</thead>
<tbody>
<tr>
<td>file system</td>
<td>network stack</td>
</tr>
<tr>
<td>virtual memory</td>
<td>process mgmt</td>
</tr>
<tr>
<td>• • • etc. • • •</td>
<td></td>
</tr>
</tbody>
</table>

- open(), read(), write(), close(), ...
- connect(), listen(), read(), write(), ...
- brk(), shm_open(), ...
- fork(), wait(), nice(), ...
UNIX as a protection system

- **OS isolates processes from each other**
  - but permits controlled sharing between them
    - through shared name spaces (e.g., FS names)

- **OS isolates itself from processes**
  - and therefore, must prevent processes from accessing the hardware directly

- **OS is allowed to access the hardware**
  - user-level processes run with the CPU in unprivileged mode
  - when the OS is running, the CPU is set to privileged mode
  - user-level processes invoke a system call to safely enter the OS
UNIX as a protection system

A CPU (thread of execution) is running user-level code in process A; that CPU is set to unprivileged mode.
UNIX as a protection system

code in process A invokes a system call; the hardware then sets the CPU to *privileged mode* and traps into the OS, which invokes the appropriate system call handler.
UNIX as a protection system

because the CPU executing the thread that’s in the OS is in privileged mode, it is able to use privileged instructions that interact directly with hardware devices like disks.
UNIX as a protection system

Once the OS has finished servicing the system call (which might involve long waits as it interacts with HW) it:

(a) sets the CPU back to unprivileged mode, and

(b) returns out of the system call back to the user-level code in process A.
UNIX as a protection system

the process continues executing whatever code that is next after the system call invocation
Hardware Privilege Modes

• A privilege mode is a hardware state that restricts the kinds of operations of actions that code may perform
  ‣ e.g., prevents direct access to hardware, process controls, and key instructions

• There are two modes we are principally concerned about in this class, user and kernel modes
  ‣ user mode is used for normal programs running with low privilege (also system services that run in “user space”)
  ‣ kernel mode is the operating system running
Device Drivers

• A *device driver* is a software module (program) that implements the interface to a piece of hardware (often needs kernel mode privilege)
  ‣ e.g., printers, monitors, graphics cards, USB devices, etc.
  ‣ often provided by the manufacturer of the device
  ‣ for performance reasons, the driver is commonly run within the operating system as part of the kernel (in kernel space)
  ‣ device drivers were often compiled into the kernel
    • required the administrator to re-compile the operating system when a new device type was introduced
    • each system had a different kernel
Recompiling Kernels?

• Recompilation of the kernel is problematic
  ‣ takes a long time
  ‣ requires sophistication
  ‣ versioning problems

• Solution 1
  ‣ User-space modules - creating user-space programs that support the operating system
    • leverages protection (against buggy code)
    • allows independent patching and upgrading
    • removes dependency on kernel version (mostly)
  • Problem: performance
    ‣ For high speed hardware, context switching is costly
Recompiling Kernels?

• Solution 2:
  ‣ *Kernel modules* (AKA, loadable kernel modules) - are software modules that run in kernel space that can be loaded (and unloaded) on a running system
  • thus, we can extend the kernel functionality without recompilation
  • the trick is that the kernel provides generic interfaces (APIs) that the module uses to communicate with the kernel
  • this is used by almost every modern OS (OSX, Windows, etc.)

**Tip**: if you want to see what modules are running on your UNIX system, use the “lsmod” command, e.g., “lsmod”.
Memory Hierarchies

• Some fundamental and enduring properties of hardware and software:
  ‣ Fast storage technologies cost more per byte, have less capacity, and require more power (heat!).
  ‣ The gap between CPU and main memory speed is widening.
  ‣ Well-written programs tend to exhibit good locality.

• These fundamental properties complement each other beautifully.

• They suggest an approach for organizing memory and storage systems known as a memory hierarchy.
An Example Memory Hierarchy

- **L0:** CPU registers hold words retrieved from L1 cache
- **L1:** L1 cache (SRAM) holds cache lines retrieved from L2 cache
- **L2:** L2 cache (SRAM) holds cache lines retrieved from main memory
- **L3:** Main memory (DRAM) holds disk blocks retrieved from local disks
- **L4:** Local secondary storage (local disks) hold files retrieved from disks on remote network servers
- **L5:** Remote secondary storage (tapes, distributed file systems, Web servers)

- Smaller, faster, costlier per byte
- Larger, slower, cheaper per byte
Caches

- **Cache**: A smaller, faster storage device that acts as a staging area for a subset of the data in a larger, slower device.

- Fundamental idea of a memory hierarchy:
  - For each $k$, the faster, smaller device at level $k$ serves as a cache for the larger, slower device at level $k+1$.

- Why do memory hierarchies work?
  - Because of locality, programs tend to access the data at level $k$ more often than they access the data at level $k+1$.
  - Thus, the storage at level $k+1$ can be slower, and thus larger and cheaper per bit.

- **Big Idea**: The memory hierarchy creates a large pool of storage that costs as much as the cheap storage near the bottom, but that serves data to programs at the rate of the fast storage near the top.
General Cache Concepts

Cache

Memory

Smaller, faster, more expensive memory caches a subset of the blocks

Larger, slower, cheaper memory viewed as partitioned into “blocks”
General Cache Concepts

Cache

Memory

Smaller, faster, more expensive memory caches a subset of the blocks

Data is copied in block-sized transfer units

Larger, slower, cheaper memory viewed as partitioned into “blocks”
General Cache Concepts

Cache

8  9  14  3

Memory

0  1  2  3

4  5  6  7

8  9  10  11

12 13 14 15

Larger, slower, cheaper memory viewed as partitioned into “blocks”

Smaller, faster, more expensive memory caches a subset of the blocks

Data is copied in block-sized transfer units
General Cache Concepts

Smaller, faster, more expensive memory caches a subset of the blocks.

Data is copied in block-sized transfer units.

Larger, slower, cheaper memory viewed as partitioned into “blocks.”

Cache

Memory
General Cache Concepts

Cache

Memory

Smaller, faster, more expensive memory caches a subset of the blocks

Larger, slower, cheaper memory viewed as partitioned into “blocks”

Data is copied in block-sized transfer units

Cache

Memory
General Cache Concepts

Cache

Data is copied in block-sized transfer units

Smaller, faster, more expensive memory caches a subset of the blocks

Larger, slower, cheaper memory viewed as partitioned into “blocks”

Memory
General Cache Concepts

Smaller, faster, more expensive memory (cache) caches a subset of the blocks.

Data is copied in block-sized transfer units.

Larger, slower, cheaper memory viewed as partitioned into “blocks”.

Cache:

```
  4  9  14  3
```

Memory:

```
  0  1  2  3
  4  5  6  7
  8  9 10 11
 12 13 14 15
```

CMPSC 311 - Introduction to Systems Programming

Page 23
General Cache Concepts

Smaller, faster, more expensive memory caches a subset of the blocks.

Data is copied in block-sized transfer units.

Larger, slower, cheaper memory viewed as partitioned into "blocks".

Cache

Memory
Cache Hit

Cache

Memory
Cache Hit

Request: 14

Data in block b is needed

Cache

Memory

Data in block b is needed
Cache Hit

Request: 14

Data in block b is needed

Block b is in cache:
Hit!
Cache Miss

Cache

| 8 | 9 | 14 | 3 |

Memory

| 0 | 1 | 2 | 3 |
| 4 | 5 | 6 | 7 |
| 8 | 9 | 10 | 11 |
| 12 | 13 | 14 | 15 |

......
Cache Miss

Request: 12

Data in block b is needed

Cache

Memory

Data in block b is needed
Cache Miss

Data in block b is needed

Block b is not in cache: Miss!

Request: 12

Cache

Memory
Cache Miss

Data in block b is needed

Block b is not in cache: **Miss**!

Block b is fetched from memory
Cache Miss

Data in block b is needed

Block b is not in cache: Miss!

Block b is fetched from memory
Cache Miss

Data in block b is needed

Block b is not in cache: Miss!

Block b is fetched from memory
Cache Miss

Data in block b is needed

Block b is not in cache: Miss!

Block b is fetched from memory

Block b is stored in cache

- Placement policy: determines where b goes
- Replacement policy: determines which block gets evicted (victim)
Types of Cache Misses

- **Cold (compulsory) miss**
  - Cold misses occur because the cache is empty.

- **Conflict miss**
  - Most caches limit blocks at level k+1 to a small subset (sometimes a singleton) of the block positions at level k.
    - E.g. Block i at level k+1 must be placed in block (i mod 4) at level k.
  - Conflict misses occur when the level k cache is large enough, but multiple data objects all map to the same level k block.
    - E.g. Referencing blocks 0, 8, 0, 8, 0, 8, ... would miss every time.

- **Capacity miss**
  - Occurs when the set of active cache blocks (working set) is larger than the cache.
Conflict Miss

Level K+1

<table>
<thead>
<tr>
<th></th>
<th>X</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>X</td>
<td>Y</td>
</tr>
<tr>
<td>101</td>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>110</td>
<td>G</td>
<td>K</td>
</tr>
</tbody>
</table>

Level K

<table>
<thead>
<tr>
<th></th>
<th>X</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>X</td>
</tr>
<tr>
<td>101</td>
<td>-</td>
</tr>
<tr>
<td>110</td>
<td>K</td>
</tr>
</tbody>
</table>
Conflict Miss
Conflict Miss

```
Level K+1
100  X  Y
101  A  B
110  G  K

Level K
100  X
101  A
110  K
```

(REQ A)
Conflict Miss
Conflict Miss

(REQ B)

Level K+1

<table>
<thead>
<tr>
<th>100</th>
<th>X</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>110</td>
<td>G</td>
<td>K</td>
</tr>
</tbody>
</table>

Level K

<table>
<thead>
<tr>
<th>100</th>
<th>X</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>A</td>
</tr>
<tr>
<td>110</td>
<td>K</td>
</tr>
</tbody>
</table>
Conflict Miss
## Conflict Miss

<table>
<thead>
<tr>
<th>Level K+1</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>X</td>
<td>Y</td>
</tr>
<tr>
<td>101</td>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>110</td>
<td>G</td>
<td>K</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Level K</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>101</td>
<td></td>
<td>B</td>
</tr>
<tr>
<td>110</td>
<td></td>
<td>K</td>
</tr>
</tbody>
</table>
Cache replacement

• When your cache is full and you acquire a new value, you must evict a previously stored value
  ‣ Performance of cache is determined by how smart you are in evicting values, known as a *cache eviction policy*
  ‣ Popular policies
    • *Least recently used* (LRU) - eject the value that has been in the cache the longest without being accessed
    • *Least frequently used* (LFU) - eject the value that accessed the least number of times
    • *First in-first out* (FIFO) - eject the same order they come in
  ‣ Policy efficiency is measured by the hit performance (how often is something asked for and found) and measured costs
    • Determined by *working set* and *workload*
## Caching in the Hierarchy

<table>
<thead>
<tr>
<th>Cache Type</th>
<th>What is Cached?</th>
<th>Where is it Cached?</th>
<th>Latency (cycles)</th>
<th>Managed By</th>
</tr>
</thead>
<tbody>
<tr>
<td>Registers</td>
<td>4-8 bytes words</td>
<td>CPU core</td>
<td>0</td>
<td>Compiler</td>
</tr>
<tr>
<td>TLB</td>
<td>Address translations</td>
<td>On-Chip TLB</td>
<td>0</td>
<td>Hardware</td>
</tr>
<tr>
<td>L1 cache</td>
<td>64-bytes block</td>
<td>On-Chip L1</td>
<td>1</td>
<td>Hardware</td>
</tr>
<tr>
<td>L2 cache</td>
<td>64-bytes block</td>
<td>On/Off-Chip L2</td>
<td>10</td>
<td>Hardware</td>
</tr>
<tr>
<td>Virtual Memory</td>
<td>4-KB page</td>
<td>Main memory</td>
<td>100</td>
<td>Hardware + OS</td>
</tr>
<tr>
<td>Buffer cache</td>
<td>Parts of files</td>
<td>Main memory</td>
<td>100</td>
<td>OS</td>
</tr>
<tr>
<td>Disk cache</td>
<td>Disk sectors</td>
<td>Disk controller</td>
<td>100,000</td>
<td>Disk firmware</td>
</tr>
<tr>
<td>Network buffer cache</td>
<td>Parts of files</td>
<td>Local disk</td>
<td>10,000,000</td>
<td>AFS/NFS client</td>
</tr>
<tr>
<td>Browser cache</td>
<td>Web pages</td>
<td>Local disk</td>
<td>10,000,000</td>
<td>Web browser</td>
</tr>
<tr>
<td>Web cache</td>
<td>Web pages</td>
<td>Remote server disks</td>
<td>1,000,000,000</td>
<td>Web proxy server</td>
</tr>
</tbody>
</table>
Next time

• We have a guest lecturer to speak to the basics of UNIX and how to make it work.