Overview of Operating Systems

CS201 Lecture 1

Jason Hibbeler
University of Vermont
Fall 2019

Topic Outline

- Complete and total overview an operating system
- Processes
- Memory and protection
- Storage

Basic Role of an OS

Basic mission:

- manage resources for users of the system.

What are the resources?

Basic Role of an OS

Basic mission:

- manage resources for users of the system

What are the resources?

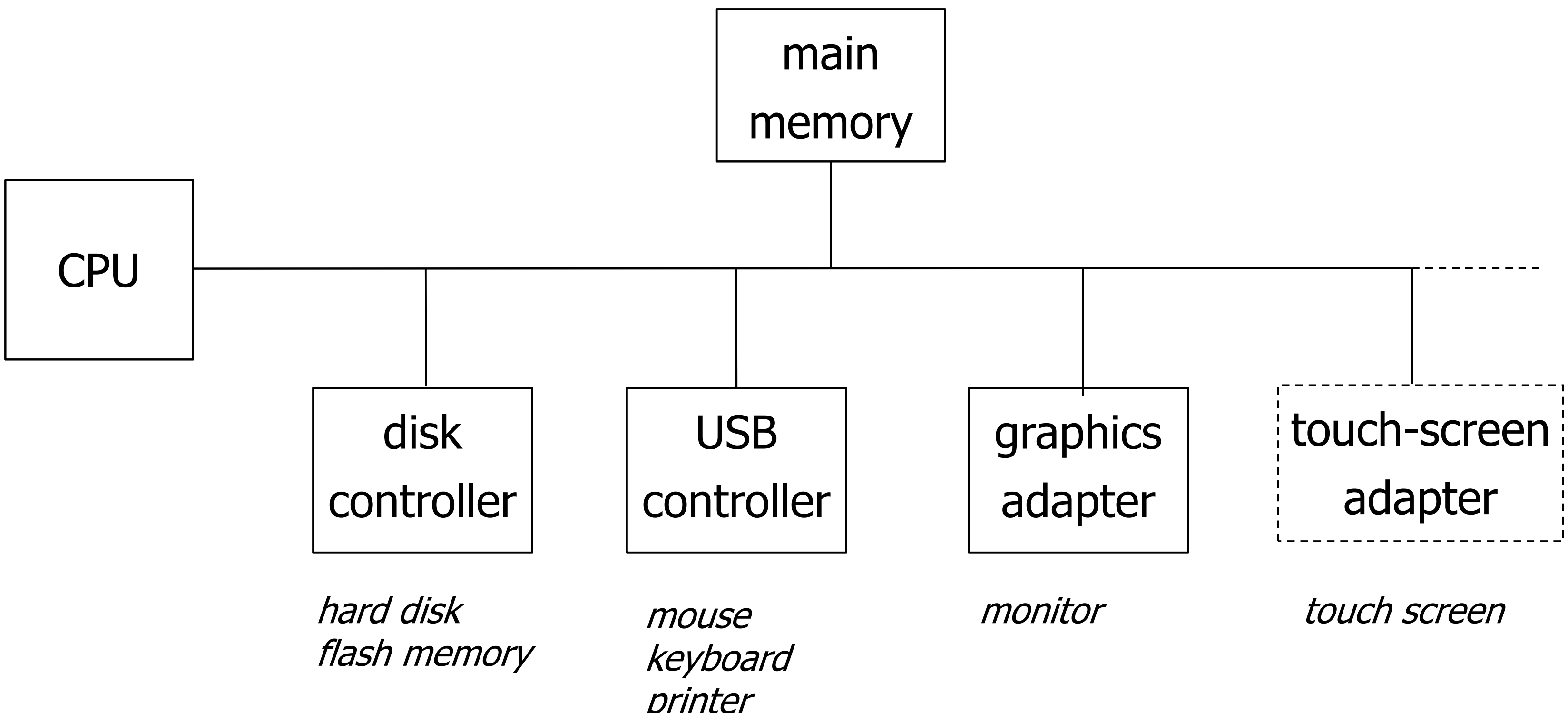
- CPU, memory, storage, I/O devices

Who wants/needs to use these resources?

The System

What is a system?

- PC, Mac, phone, iPhone, supercomputer, mainframe, RPi, etc.



Operating System

What is an operating system?

- it's the one program that is always running on the system
- the core part is called "the kernel"
- controls the hardware
- facilitates the execution of user programs
- also has other auxiliary programs to help out

the Colonel

System Initialization

Turn on the power, and…

- the system loads the "bootstrap image" from ROM (read-only memory)
- this bootstrap image is also called the firmware

The bootstrap image then loads the OS

- and starts various auxiliary processes ("daemons", for Unix and macOS, "services" for Windows)

The OS in Operation

Wait for something to happen:

- wait for an event
- then handle the event
- an event is usually signaled by an interrupt—kind of a low-level "hello...I need attention!"

Interrupt Handling

When something happens, take care of it

- the CPU stops what it was doing and transfers to a special location: the interrupt service code for that interrupt
- this operation must be fast!
- it's implemented in hardware by loading an interrupt vector: an array of addresses indexed by unique device number

Interrupt Handling

Must be fast!

bang! *interrupt comes in on a special dedicated line*

processing ... processing ... processing ... processing *processing ... processing ... processing ... processing*

interrupt #1	address A1
interrupt #2	address A2
interrupt #3	address A3

⋮

CPU jumps to interrupt-handler routine

- looks up address of code to handle that specific interrupt
- saves state associated with current task
- jumps to the specified address
- runs that code
- returns and restores saved state

Storage

Main memory

- random-access memory (RAM)
- sometimes it's SRAM, sometimes DRAM
- key attributes are that it's limited in size
- and it's volatile (turn off power $\Rightarrow$ lose contents of memory)

Memory holds program instructions and program data

Also, there is read-only memory (ROM, EEPROM)

By the way, what is a byte? What is a "megabyte"?
What is a "gigabyte"?

Memory Hierarchy

How long is a CPU cycle?

on-chip registers (one CPU cycle)

on-chip cache (a few CPU cycles—hundreds of GB per second)

main memory (10 GB per second)

solid-state disk (1-2 GB per second)

hard disk drive (10 MB per second)

optical disk (< 10 MB per second)

magnetic tape (slow)

the CPU

I/O Processing

Each hardware device has a device controller that is responsible for the actual interface to the hardware

OS has a device driver that talks to the device controller

Device driver loads registers in the device controller with instructions ("read a keystroke")

- device controller then signals to the OS, with an interrupt, that's it's done

Special Case: DMA

Direct memory access: the device controller can transfer a block of data directly to main memory

- without the intervention of the CPU
- why is this useful?

Computer Systems Architecture

In the beginning, there was the single-processor system

- one CPU, doing one thing at a time
- there are usually auxiliary specialized processors as well (disk controller, keyboard controller, etc.)

Nowadays: multiprocessor (or multicore) systems are the norm

- note that “core” and “processor” are not synonymous
- multicore systems are truly the norm now

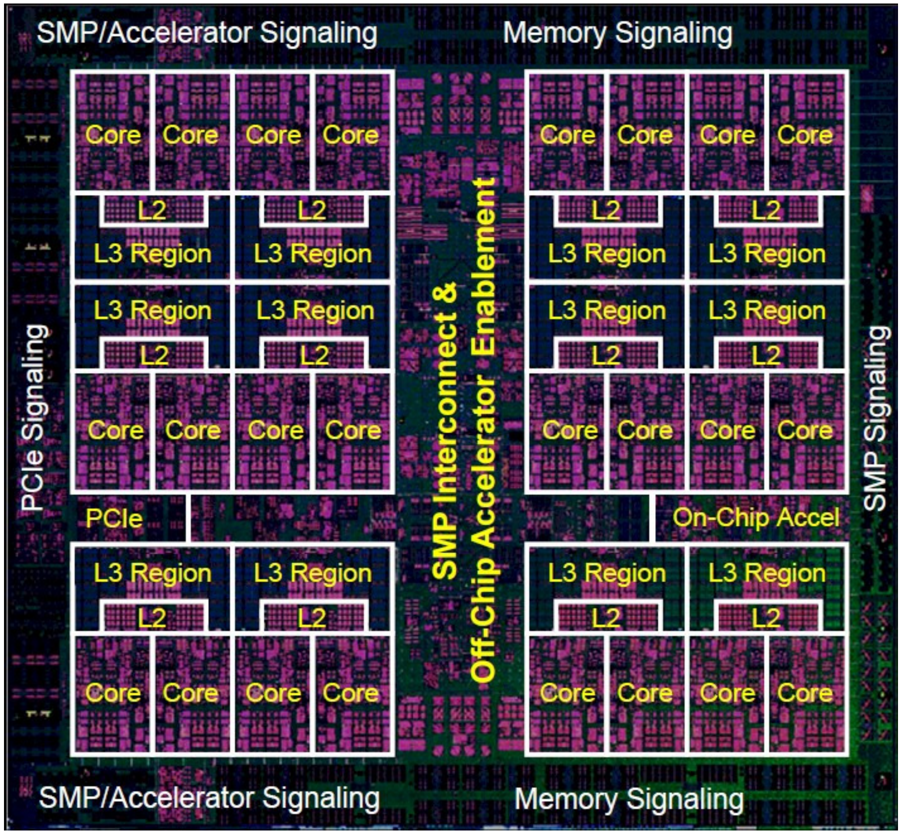
Apple A11 Bionic

The A11 features an Apple-designed 64-bit ARMv8-A six-core CPU, with two high-performance cores at 2.39 GHz, called **Monsoon**, and four energy-efficient cores, called **Mistral**. The A11 uses a new second-generation performance controller, which permits the A11 to use all six cores simultaneously, unlike its predecessor the A10. The A11 also integrates an Apple-designed three-core graphics processing unit (GPU) with 30% faster graphics performance than the A10. Embedded in the A11 is the M11 motion coprocessor. The A11 includes a new image processor which supports computational photography functions such as lighting estimation, wide color capture, and advanced pixel processing. [from Wikipedia]

TechInsights

from https://wccftech.com/apple-iphone-8-plus-a11-bionic-complete-teardown/

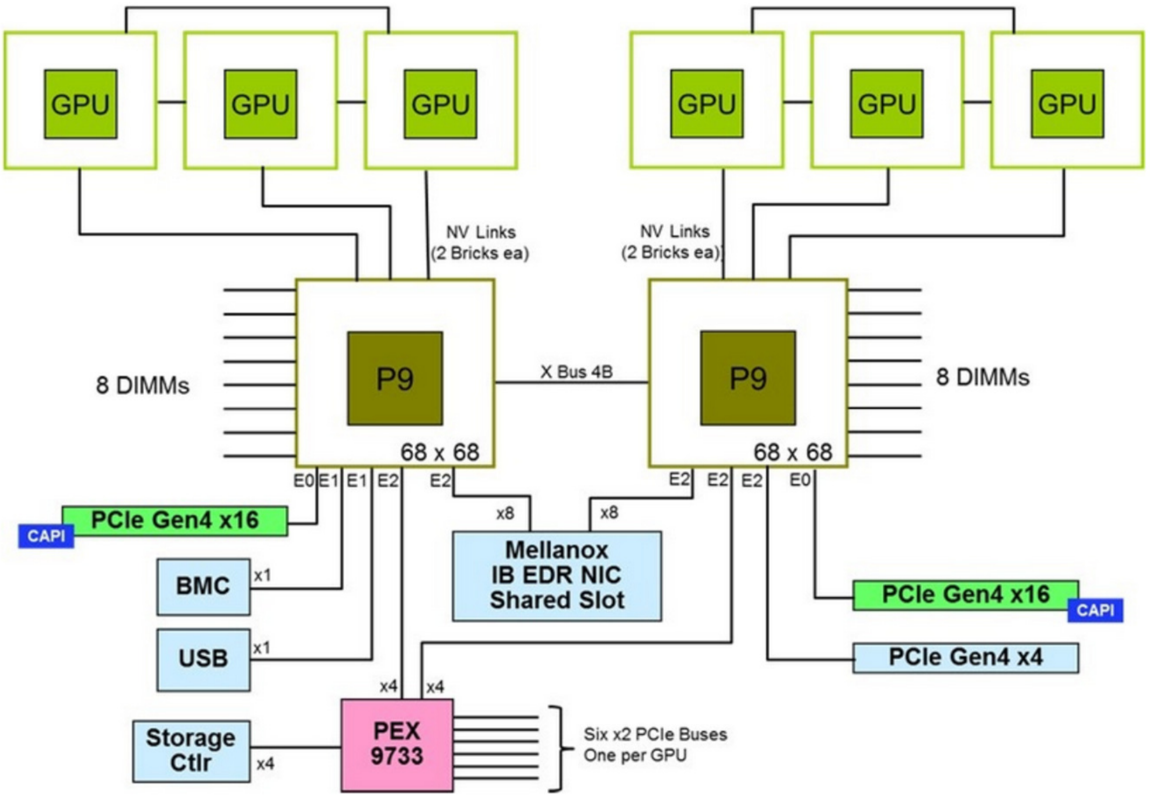
Multicore Processor



IBM POWER9 Processor

announced August 2016

POWER9 + GPU System



https://www.nextplatform.com/2017/12/05/power9-to-the-people/

Multicore Systems

Increased throughput: two CPUs can do more work in a given time than one CPU can!

Lower cost: multicore systems can share peripherals

More reliable: if one core fails, then the system can still function

Symmetric (SMP) vs. asymmetric (non-SMP) processing

- does every processor do the same thing for a particular program?
- or do different processors do different tasks?

Computer Systems Architecture

Nowadays: multiprocessor (or multicore) systems are the norm

- note that "core" and "processor" are not synonymous
- multicore systems are truly the norm now

Specialized Processors

GPU: graphics processing unit

- highly parallel hardware engine
- very good at solving a simple task efficiently
- more recently: much faster at solving the computations required for neural networks

TPU: Tensor processing unit

- custom hardware engine built by Google for TensorFlow
- to run machine-learning applications

Operating System Structure

Most important characteristic: must support *multiprogramming*

- we don't want the CPU to be idle when a job is performing I/O
- in a multiprogrammed system, the OS will keep a set of jobs in a job pool (let's say this is in main memory)
- when it's turn for job A to run, the CPU switches its execution to A, and A starts to run
- when that job makes a request for I/O service, the OS switches to a different task – task B
- and so on
- basic goal: prevent the CPU from being idle (refer to the picture on Slide #11)

Timesharing

Natural extension of multiprogramming is *timesharing*: enable many users to share system resources (most notably, the CPU)

- system switches rapidly between different users, giving them each a short time slice
- this gives all the users the illusion that they each have a dedicated machine!
- this will work effectively only if the response time for each user is reasonable and consistent
- this requires the OS to perform complex management of resources and requests
- *process*: a job that is loaded into memory and is ready for execution

Memory Management

physical memory

- the actual memory (the bits and bytes) on the machine

virtual memory

- the view of memory that the OS uses and which a process sees
- can be much larger than physical memory
- the OS translates virtual-memory references to physical-memory references

OS Operation

Simplest description of what the OS does:

- sit and wait for something to happen
- when something happens, take the appropriate action

Slightly more sophisticated description:

- let a process run until it makes an I/O request, or until timeout occurs, or until an unexpected condition in the software occurs*
- then, give the CPU to a different process

* what might this be? what's an unexpected condition that could occur in your software?

Dual-Mode Operation

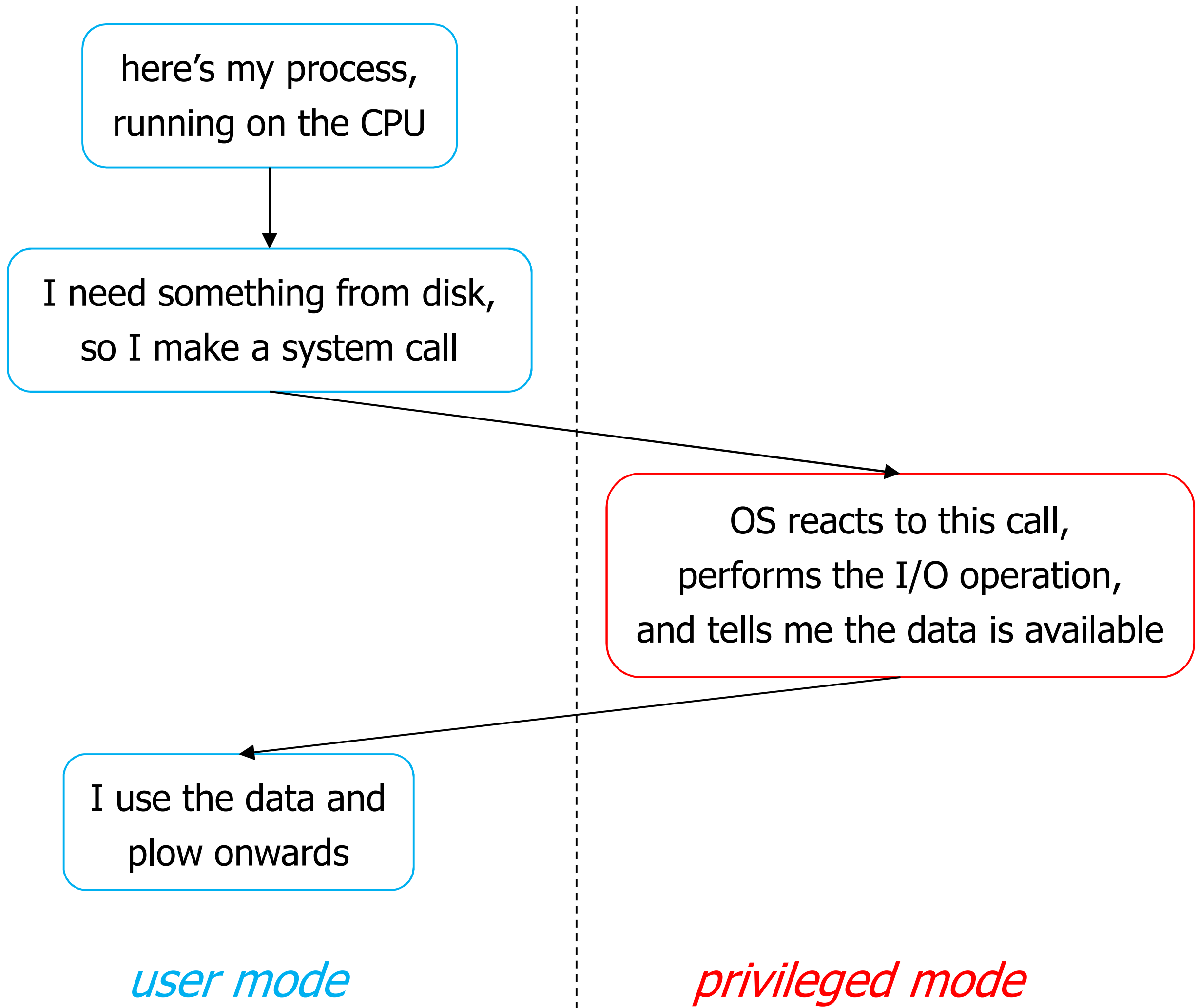
Multiprogrammed system: many different jobs and users, all using the same resources

- need to prevent user A from accessing user B's memory!
- need to prevent user A from reading user B's private data!

This leads to the concept of dual-mode operation

- the OS needs special powers to facilitate this protection
- there are two modes of operation: user mode and kernel mode (aka privileged mode, system mode, supervisor mode)
- the mode is enforced in the hardware itself through a mode bit

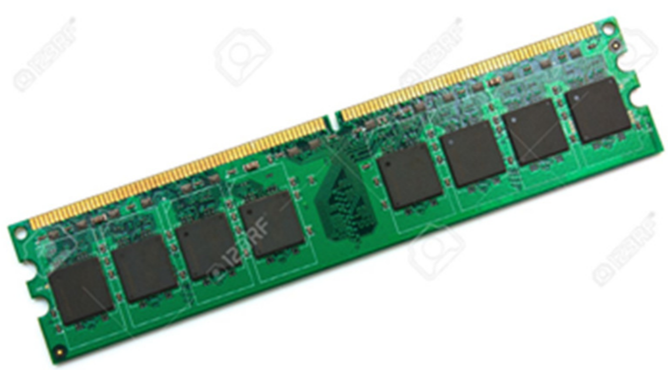
Dual-Mode Operation

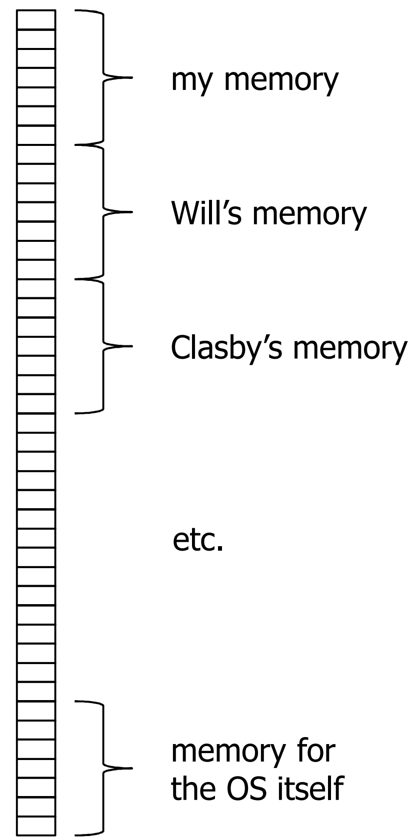


Memory: Extremely Simplified View of Memory Management

Think of the physical memory in the system

- the physical memory is the collection of memory boards in the system
- there's a finite amount of memory, and so the OS has to divvy the memory up among all the current users (actually, the current processes) on the machine





Protection

If a process attempts an operation for which it does not have permission, the hardware intercedes!

- if I try to read memory that is not in my address space, I get a hardware exception

```
#include <stdio.h>

int main(int argc, char** argv) {
  int *ip;
  int dummy, idx;

  printf("hello!\n");

  for (idx = 0; idx < 1024; ++idx) {
    printf("idx = %d\n", idx);
    dummy = ip[1024 * idx];
  }
}
```

Protection

Compiling and running this program:

```
squall|/users/j/h/jhibbele/CS201
> gcc -g sigsegv.c
squall|/users/j/h/jhibbele/CS201
> ./a.out
hello!
idx = 0
idx = 1
idx = 2
idx = 3
Segmentation fault
squall|/users/j/h/jhibbele/CS201
>
```

What happens if I do this in Python?

What happens if I do this in Java?

The Timer

OS must prevent a user process from keeping the CPU too long (when could this happen)?

This OS maintains a timer:

- when a user job gets the CPU, set the timer to t units
- when the timer expires, interrupt the job (if it still has the CPU)
- and at this point, a different job will be able to use the CPU

Question: who is permitted to modify the timer?

A Process

Program: sequence of operations

Process: a program in execution

- process needs resources (CPU, I/O, memory), and it's the job of the OS to assign these resources to the process
- CPU executes one instruction at a time for a process
- when the process terminates, the OS cleans up after the process
- some processes belong to the OS itself
- a process can have several threads of execution (a thread is kind of like a piece of a process)

Snapshot of Processes

```
root         1     0  0 Jan02 ?        00:01:00 /sbin/init
root         2     0  0 Jan02 ?        00:00:00 [kthreadd]
root         3     2  0 Jan02 ?        00:00:00 [migration/0]
root         4     2  0 Jan02 ?        00:00:49 [ksoftirqd/0]
root         5     2  0 Jan02 ?        00:00:00 [stopper/0]
root         6     2  0 Jan02 ?        00:00:20 [watchdog/0]
root         7     2  0 Jan02 ?        01:33:13 [events/0]
root         8     2  0 Jan02 ?        00:00:00 [events/0]
root         9     2  0 Jan02 ?        00:00:00 [events_long/0]
root        10     2  0 Jan02 ?        00:00:00 [events_power_ef]
root        11     2  0 Jan02 ?        00:00:00 [cgroup]
root        12     2  0 Jan02 ?        00:00:02 [khelper]
root        13     2  0 Jan02 ?        00:00:00 [netns]
root        14     2  0 Jan02 ?        00:00:00 [async/mgr]
root        15     2  0 Jan02 ?        00:00:00 [pm]
root        16     2  0 Jan02 ?        00:00:57 [sync_supers]
root        17     2  0 Jan02 ?        00:00:02 [bdi-default]
root        18     2  0 Jan02 ?        00:00:00 [kintegrityd/0]
root        19     2  0 Jan02 ?        00:02:33 [kblockd/0]
root        20     2  0 Jan02 ?        00:00:00 [kacpid]
root        21     2  0 Jan02 ?        00:00:00 [kacpi_notify]
root        22     2  0 Jan02 ?        00:00:00 [kacpi_hotplug]
root        23     2  0 Jan02 ?        00:00:00 [ata_aux]
root        24     2  0 Jan02 ?        00:00:00 [ata_sff/0]
root        25     2  0 Jan02 ?        00:00:00 [ksuspend_usbd]
root        26     2  0 Jan02 ?        00:00:00 [khubd]
root        27     2  0 Jan02 ?        00:00:00 [kseriod]
root        28     2  0 Jan02 ?        00:00:00 [md/0]
root        29     2  0 Jan02 ?        00:00:00 [md_misc/0]
root        30     2  0 Jan02 ?        00:00:00 [linkwatch]
root        33     2  0 Jan02 ?        00:00:04 [khungtaskd]
root        34     2  0 Jan02 ?        00:00:16 [kswapd0]
root        35     2  0 Jan02 ?        00:00:00 [ksmd]
root        36     2  0 Jan02 ?        00:01:56 [khugepaged]
root        37     2  0 Jan02 ?        00:00:00 [aio/0]
root        38     2  0 Jan02 ?        00:00:00 [crypto/0]
root        45     2  0 Jan02 ?        00:00:00 [kthrotld/0]
root        46     2  0 Jan02 ?        00:00:00 [pciehpd]
root        48     2  0 Jan02 ?        00:00:00 [kpsmoused]
root        49     2  0 Jan02 ?        00:00:00 [usbhid_resumer]
root        50     2  0 Jan02 ?        00:00:00 [deferwq]
root        82     2  0 Jan02 ?        00:00:00 [kdmremove]
root        83     2  0 Jan02 ?        00:00:00 [kstriped]
root       112     2  0 Jan02 ?        00:00:00 [ttm_swap]
root       209     2  0 Jan02 ?        00:00:00 [scsi_eh_0]
root       210     2  0 Jan02 ?        00:00:00 [scsi_eh_1]
root       216     2  0 Jan02 ?        00:00:00 [scsi_eh_2]
root       217     2  0 Jan02 ?        00:00:00 [vmw_pvscsi_wq_2]
root       341     2  0 Jan02 ?        00:00:00 [kdmflush]
root       343     2  0 Jan02 ?        00:00:00 [kdmflush]
root       361     2  0 Jan02 ?        00:04:59 [jbd2/dm-0-8]
root       362     2  0 Jan02 ?        00:00:00 [ext4-dio-unwrit]
root       437     1  0 Jan02 ?        00:00:18 /sbin/udevd -d
root       577     2  0 Jan02 ?        00:04:28 [vmmemctl]
root       748     2  0 Jan02 ?        00:00:00 [jbd2/sda1-8]
root       749     2  0 Jan02 ?        00:00:00 [ext4-dio-unwrit]
root       838     2  0 Jan02 ?        00:01:02 [kauditd]
root      1358     2  0 Jan02 ?        00:04:27 [flush-253:0]
root      1418     1  0 Jan02 ?        00:05:07 auditd
nslcd     1452     1  0 Jan02 ?        00:34:28 /usr/sbin/nslcd
root      1468     1  0 Jan02 ?        00:07:09 /sbin/rsyslogd -i /var/run/syslogd.pid -c 5
dbus      1539     1  0 Jan02 ?        00:00:00 dbus-daemon --system
root      1591     2  0 Jan02 ?        00:04:12 [rpciod/0]
root      1593     2  0 Jan02 ?        00:00:00 [kslowd000]
root      1594     2  0 Jan02 ?        00:00:00 [kslowd001]
root      1595     2  0 Jan02 ?        00:00:12 [nfsiod]
root      1596     2  0 Jan02 ?        00:00:00 [nfsv4.0-svc]
root      1738     1  0 Jan02 ?        00:05:29 sendmail: accepting connections
smmsp     1747     1  0 Jan02 ?        00:00:01 sendmail: Queue runner@01:00:00 for /var/spool/clientmqueue
root      1767     1  0 Jan02 ?        00:00:00 /usr/bin/svnserve --daemon --pid-file=/var/run/svnserve.pid -d -r /usr/local/subversion
root      1779     1  0 Jan02 ?        00:03:34 crond
root      1836     1  0 Jan02 ?        00:00:00 /usr/sbin/atd
root      1849     1  0 Jan02 tty1     00:00:00 /sbin/mingetty /dev/tty1
root      1851     1  0 Jan02 tty2     00:00:00 /sbin/mingetty /dev/tty2
root      1853     1  0 Jan02 tty3     00:00:00 /sbin/mingetty /dev/tty3
root      1855     1  0 Jan02 tty4     00:00:00 /sbin/mingetty /dev/tty4
root      1859     1  0 Jan02 tty5     00:00:00 /sbin/mingetty /dev/tty5
root      1861     1  0 Jan02 tty6     00:00:00 /sbin/mingetty /dev/tty6
root      3036     1  2 11:43 ?        00:00:07 /usr/bin/python /usr/bin/fail2ban-server -b -s /var/run/fail2ban/fail2ban.sock -p /var/run/fail2ban/fail2ban.pid -x
jhibbele  4072 27208  0 11:48 pts/1    00:00:00 ps -ef
root      7636     1  0 Jan08 ?        00:00:00 cupsd -C /etc/cups/cupsd.conf
root      7670     1  0 Jan08 ?        00:03:51 /usr/sbin/sshd
root      7738     1  0 Jan08 ?        00:02:17 automount --pid-file /var/run/autofs.pid
rpc       7805     1  0 Jan08 ?        00:00:22 rpcbind -w
root      9029     1  0 Jan29 ?        00:00:00 /usr/lib/vmware-vgauth/VGAuthService -s
root     11243     1  0 Jan29 ?        02:22:48 /usr/sbin/vmtoolsd
ntp      11355     1  0 Jan29 ?        00:00:26 ntpd -u ntp:ntp -p /var/run/ntpd.pid -g
root     13162  7670  0 Jun21 ?        00:00:00 sshd: xwu [priv]
xwu      13195 13162  0 Jun21 ?        00:00:00 sshd: xwu@pts/0
xwu      13196 13195  0 Jun21 pts/0    00:00:00 -tcsh
root     24290     1  0 Jun12 ?        00:18:34 /var/cfengine/bin/cf-execd
root     24588     1  0 Jun12 ?        00:06:26 /var/cfengine/bin/cf-monitord
root     24702     1  0 Jun12 ?        00:28:43 /var/cfengine/bin/cf-serverd
root     27195  7670  0 10:56 ?        00:00:00 sshd: jhibbele [priv]
jhibbele 27207 27195  0 10:57 ?        00:00:00 sshd: jhibbele@pts/1
jhibbele 27208 27207  0 10:57 pts/1    00:00:00 -bash
rpcuser  28929     1  0 Jan08 ?        00:00:00 rpc.statd
root     32645     2  0 08:07 ?        00:00:00 [flush-0:24]
root     32748   437  0 Jan29 ?        00:00:14 /sbin/udevd -d
root     32749   437  0 Jan29 ?        00:00:00 /sbin/udevd -d
```

Processes: What the OS Does

1. schedule processes and threads on the CPUs
2. create and delete user and system processes
3. suspend and resume processes
4. provide mechanisms for process synchronization
5. provide mechanisms for process communication

Memory Management

Remember: bit vs. byte. KB, MB, GB.

Main memory: the hardware memory that the CPU can access directly

In order to access data on disk, the data must first be brought into main memory

- note that this includes the program instructions themselves!

Memory Management

As the program executes, the memory references in the program must be mapped to real physical memory

The OS will keep several programs (and their data) in memory simultaneously

- and manage the memory and the processes

Memory: What the OS Does

1. keep track of which parts of memory are being used, and who is using them
2. decide which processes and what data to move into and out of memory
3. allocate and deallocate memory as needed

Storage Management

File: collection of related information, in some particular format

A file lives on some storage medium in the system

- let’s say on a hard disk drive

The hard disk drive itself has some particular organization to it that the OS imposes

The OS additionally imposes access control on files

Storage: What the OS Does

1. create and delete files
2. create and delete directories to organize files
3. support primitive operations for manipulating files and directories
4. map files onto secondary storage (i.e. disk drive)
5. back up files on stable storage media

Mass-Storage Management

Disks and beyond; operations are:

1. managing free space
2. allocating storage
3. scheduling disk accesses

Caching

Just a quick mention—idea is to keep the data that you need close to you (see Slide #11)

Hardware memory cache, managed by the hardware

Main memory is a cache for secondary memory; managed by the OS (secondary memory—disk drive or flash memory)

I/O Subsytem

The I/O subsystem abstracts away the details of the I/O devices themselves

Key activities of the I/O subsystem of the OS:

1. buffer, cache, and spool data from the I/O devices to/from main memory
2. provide a general device-driver interface
3. provide drivers for specific hardware devices

Protection and Security

Protection: mechanism for controlling access of processes (and users) to system resources

Security: prevention of unauthorized access to system resources

The system assigns each user a unique ID

- and then grants certain privileges to each ID
- additionally, the system can form groups of users with certain privileges associated with the group

Virtualization

Virtualization is the enablement in software of a “virtual machine”

- a self-contained environment on a host operating system that appears to be a separate machine, possibly running a different OS
- useful for running programs written for one OS on a machine that uses a different OS
- also provides the basic infrastructure for cloud computing

Distributed Systems

Distributed system

- a collection of physically distinct computer systems that are networked together
- provide users access to shared resources
- might provided a distributed file system (e.g., NFS or AFS)

Kernel Data Structures

Review: here are some key data structures that an OS uses

1. lists, stacks, queues
2. trees
3. hashmaps (dictionaries)
4. bitmaps

Computing Environments

What company is this?

Computing Environments

What company is this?

Computing Environments

What company is this?

Computing Environments

What company is this?

Computing Environments

In the old days: everyone had a “personal computer”

- connected to a network
- all the big work was done on big batch servers

Now we have laptops

And mobile systems

- tablet or “smart phone”
- does a smart phone have an OS? How about a Kindle?

Connection models

- Client/server, peer-to-peer

Cloud Computing

Public cloud, private cloud

Kinds of cloud computing

- software as a service (SaaS)
- platform as a service (PaaS)
- infrastructure as a service (IaaS)

Real-Time Embedded Systems

Usually created to accomplish very specific tasks

- might be a primitive OS
- or might be a full OS, e.g., a specialized version of Linux

Often, the #1 performance criterion is response time

- think of a robot arm in a factory
- or a self-driving car (!!!)
- for non-real-time systems, the variance of the response time is important
- for real-time systems, the maximum response time is key

Finally, Open-Source Software

Linux is an open-source system

- and there are many flavors of Linux

Darwin, the kernel of macOS, is open source

Android is essentially open source