CMPSC 311 - Introduction to Systems Programming

UNIX/Operating Systems

Professors:

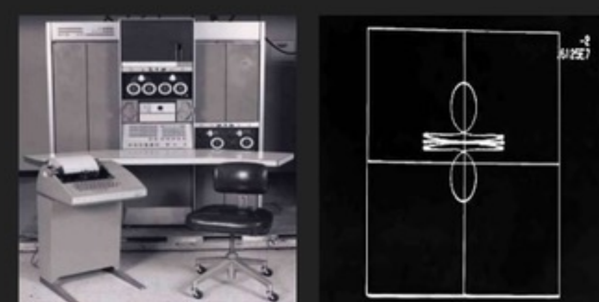
Suman Saha

(Slides are mostly by *Professor Patrick McDaniel* and *Professor Abutalib Aghayev*)

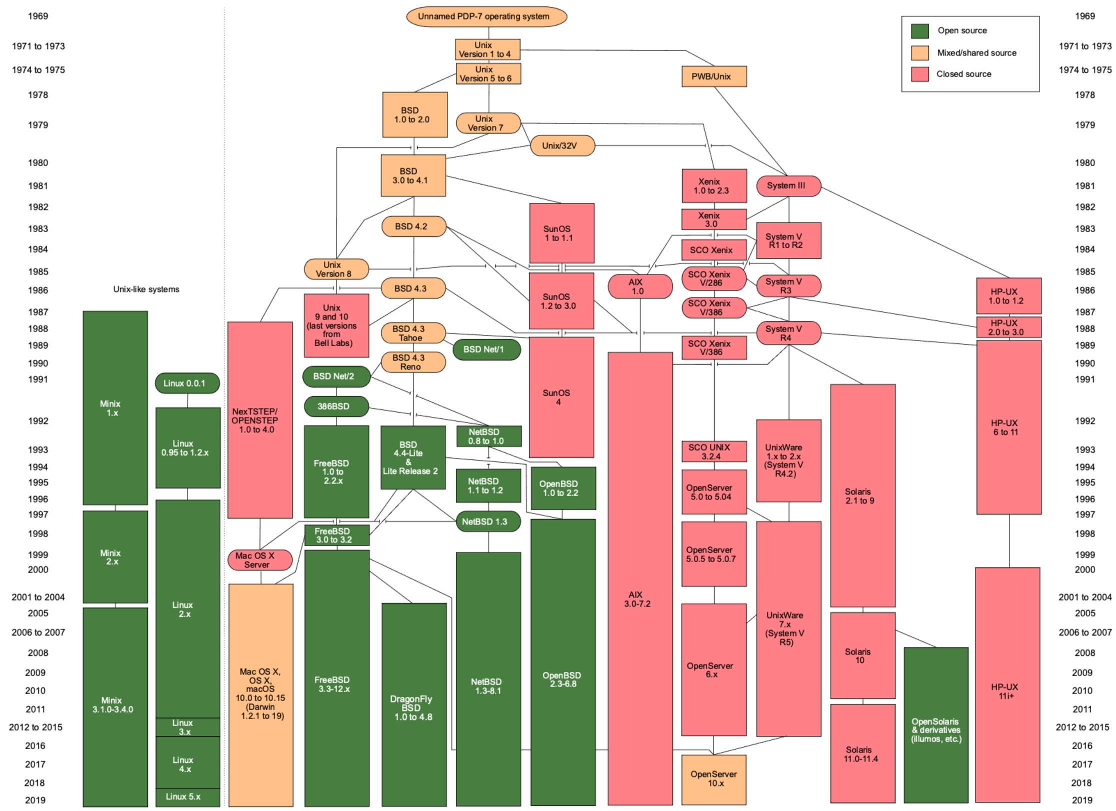
UNIX Origins

- Multics project starts in 1964
  - MIT, General Electric, Bell Labs
  - Project fails but produces many useful ideas
- Thompson and Ritchie work on Multics
  - Space Travel for Multics (GE-645)
  - Port Space Travel to PDP-7
  - Build tools, file system for PDP-7 → UNIX
- Main attributes of UNIX
  - 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.

"Space Travel"
a game for PDP-7 (1969)



UNIX Variants



Source: Wikipedia

Linux Origins

From: torvalds@klaava.Helsinki.FI (Linus Benedict Torvalds)
Subject: What would you like to see most in minix?
Date: 25 Aug 91 20:57:08 GMT

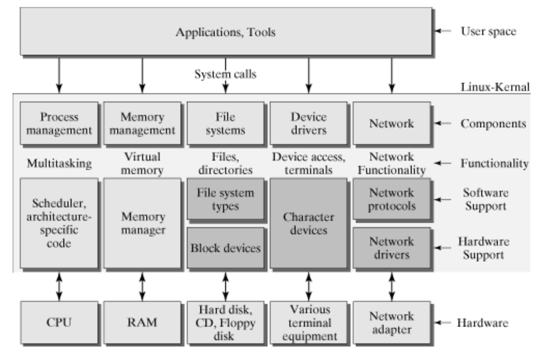
Hello everybody out there using minix -

I'm doing a (free) operating system (just a hobby, won't be big and professional like gnu) for 386(486) AT clones....

- GNU → GNU is Not UNIX
  - Free software project started by Richard Stallman in 1983
  - gcc, gdb, bash, emacs, ...
  - GNU also has an operating system kernel, Hurd – not actively developed/used
- Linux is an operating system kernel
  - Linux kernel + user tools, desktop environment, ... → Linux distribution

Linux

- Linux can be viewed as software layers
  - OS kernel -- direct interaction with hardware/firmware
  - 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 (in terminal program)
  - 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, ...)



Linux distributions

- Semi-Commercial systems
- Since 1991
  - Red Hat, SUSE/Novell, Caldera (defunct, SCO), Debian,
  - Mandrake/Mandriva, Slackware, Gentoo, Ubuntu, Knoppix, Fedora, etc., etc.
  - distrowatch.com
  - List_of_Linux_distributions (Wikipedia)

- Android, since 2003

Open source

- Many software 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)?

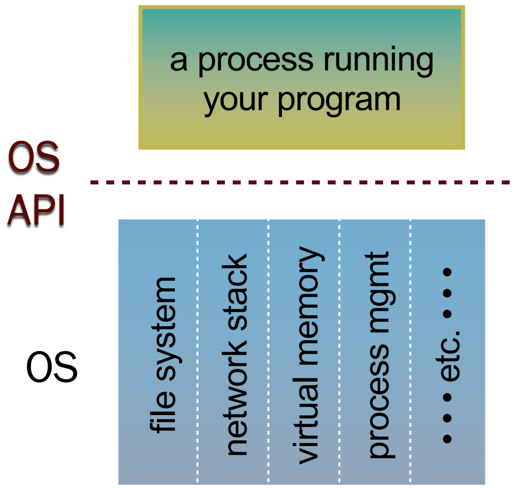
Operating Systems

- 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

Reality: an operating system is just another program, but it runs directly on the hardware ….

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 POSIX API)



file system

- open( ), read( ), write( ), close( ), ...

network stack

- connect( ), listen( ), read( ), write ( ), ...

virtual memory

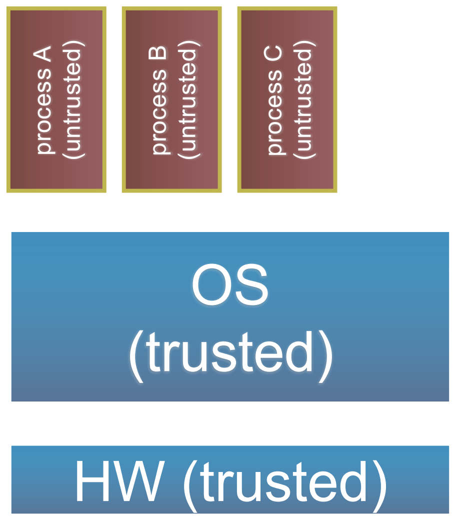
- brk( ), shm_open( ), ...

process management

- fork( ), wait( ), nice( ), ...

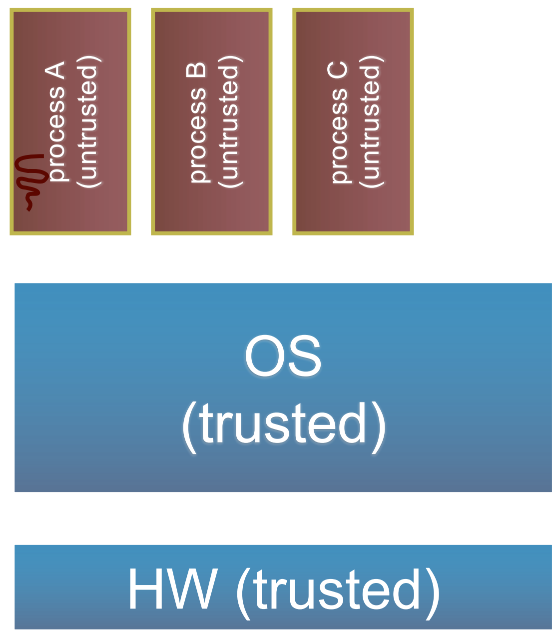
UNIX as a protection system

- 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
  - when user processes run, the CPU in unprivileged (user) mode
  - when the OS is running, the CPU is in privileged (kernel) mode
  - user-level processes invoke a system call to safely enter the OS



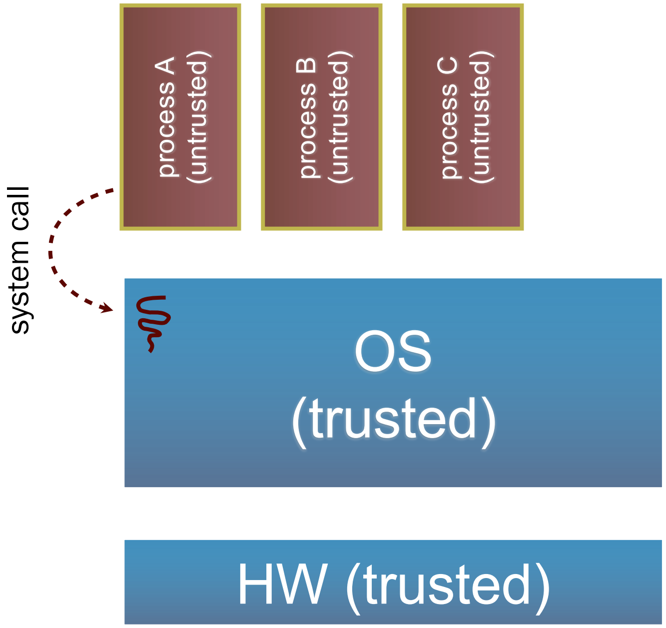
UNIX as a protection system

a CPU (thread of execution) is running user-level code in process A; that CPU is set to *unprivileged mode*



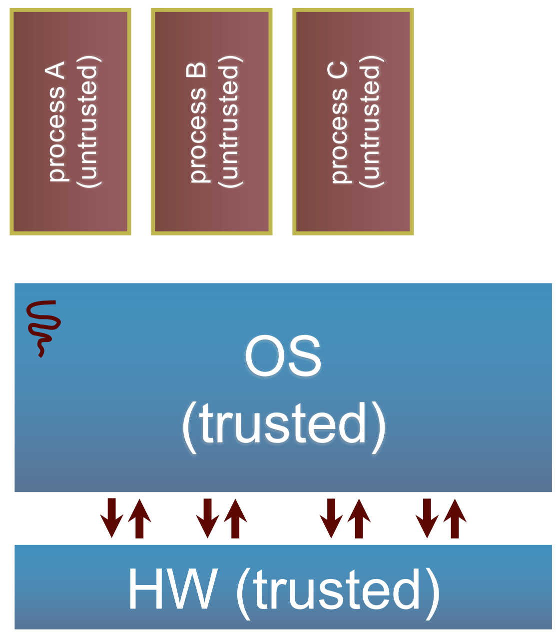
UNIX as a protection system

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

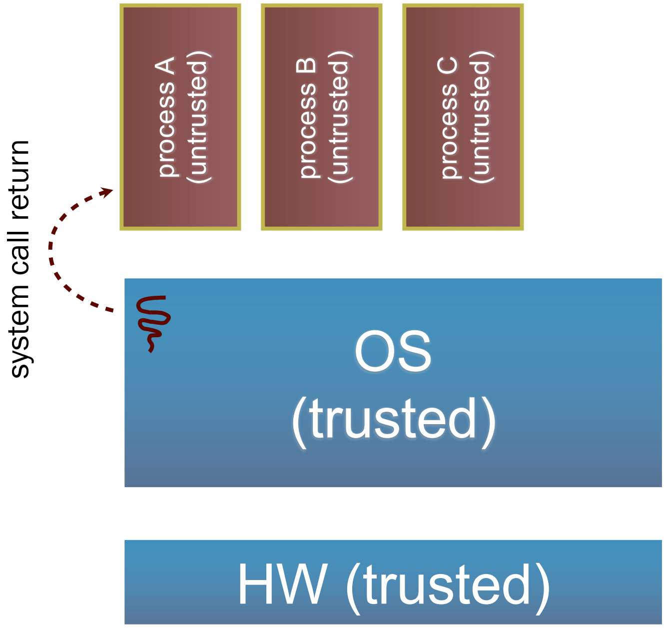


UNIX as a protection system

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

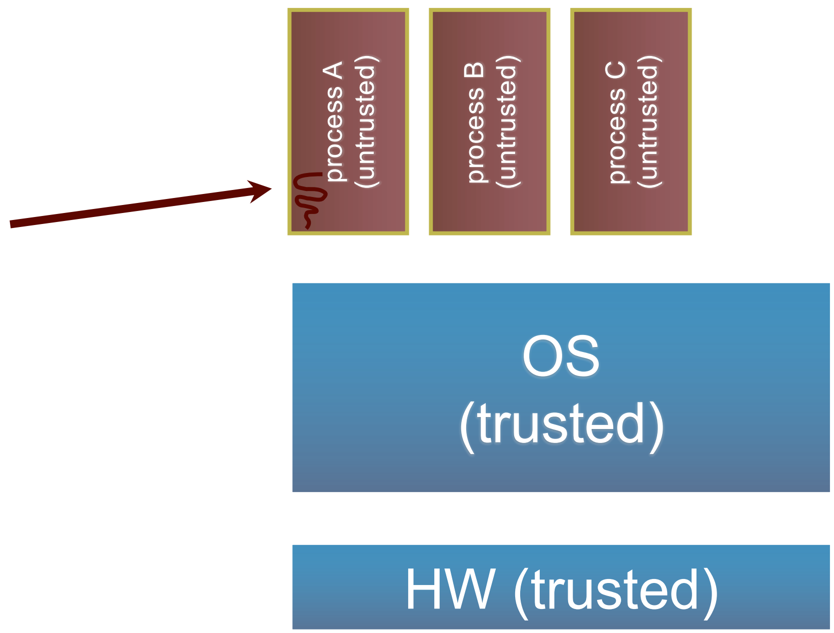
(a) sets the CPU back to unprivileged mode, and

(b) returns out of the system call back to the user-level code in process A



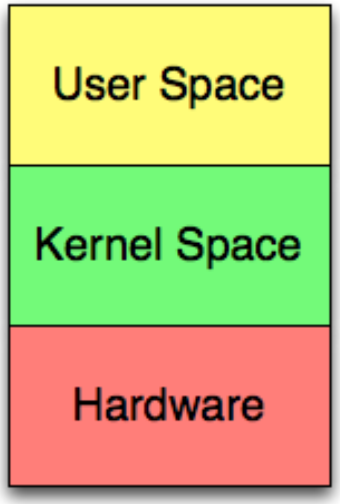
UNIX as a protection system

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



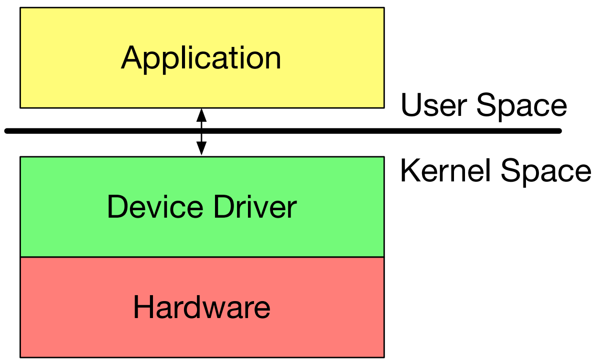
Hardware Privilege Modes

- A privilege mode is a hardware state that restricts the operations 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 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
- Unrelated to superuser (root, administrator) privileges



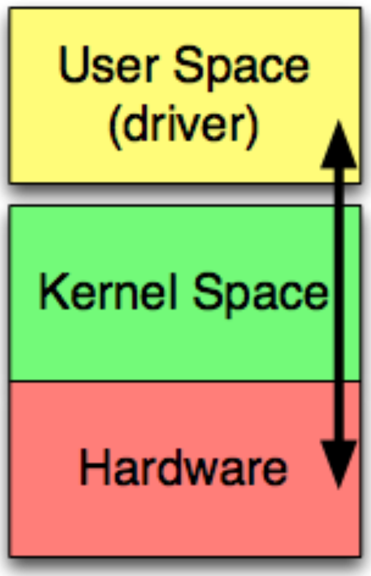
Device Drivers

- A device driver is a software module (program) that implements the interface to a piece of real or virtual 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)
  - In the past device drivers were often directly compiled into the kernel (where extensions to the operating system)
    - 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 (interacting with user space is often much slower than in-kernal operations)

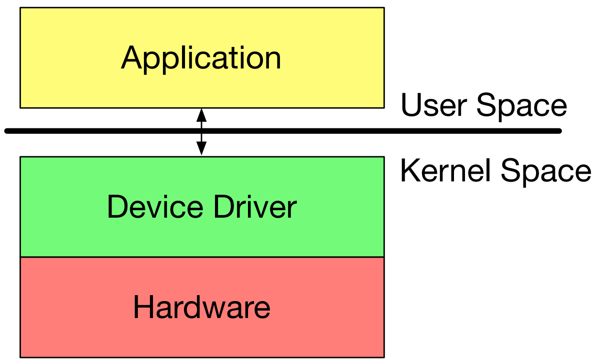


Recompiling Kernels?

- 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.