Operating Systems

Bryce Boe

2012/08/07

CS32, Summer 2012 B

Outline

- HW1 Review
- Operating Systems Overview
- Linux Software Architecture

HW1 Review

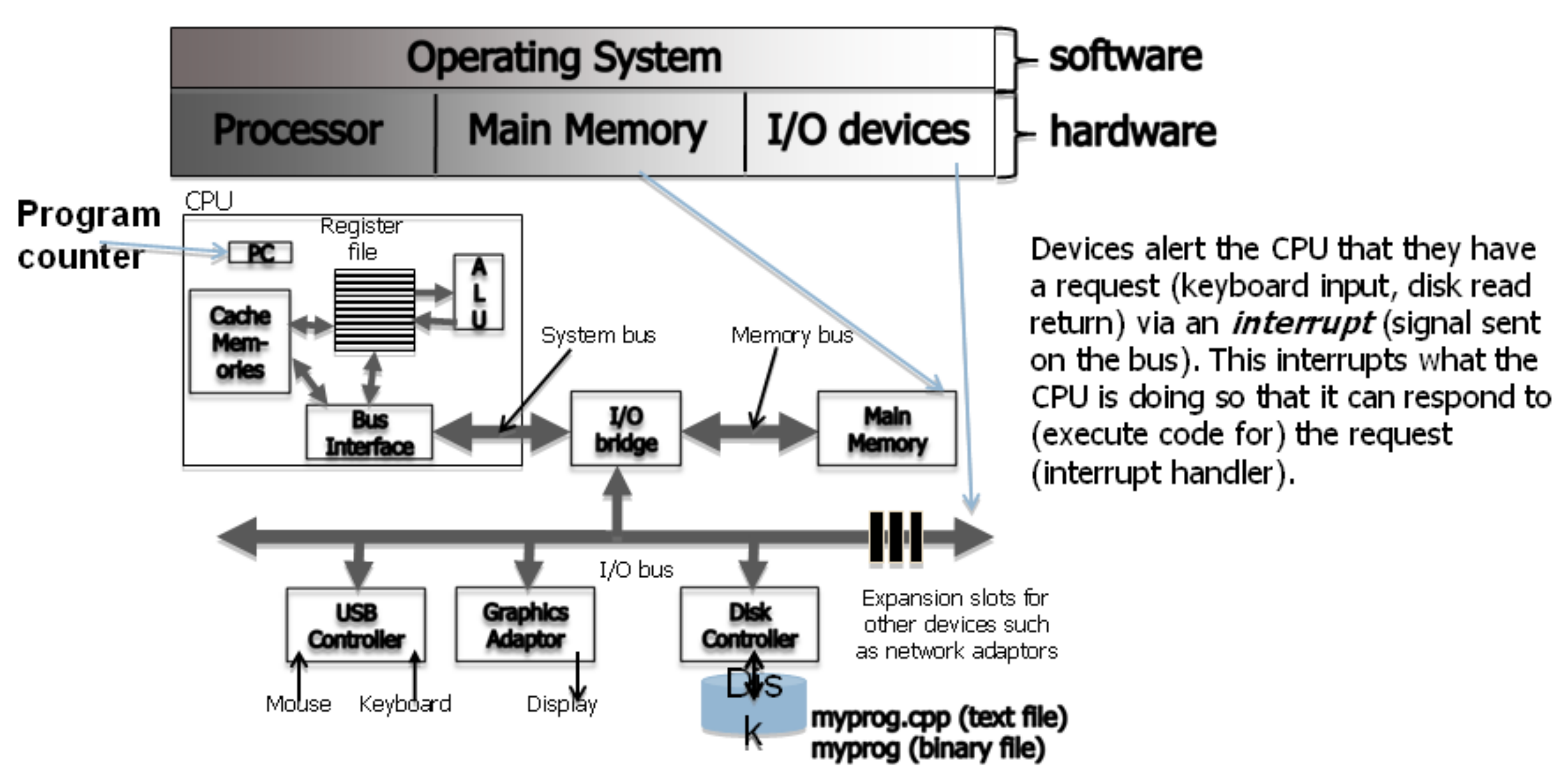
- 26 submissions, nearly all 14/14
- One issue with the grader feedback:
  - Lack of newlines at the end of the input is hard to detect, this has been corrected
- Going forward emailed submissions will not be accepted
- atoi v. stringstream example
  - See str_to_int.cpp

Operating Systems Overview

System Resources

- Central Processing Unit (CPU)
- Main memory, aka random access memory (RAM)
- Input/Output (I/O) devices
  - Keyboard, mouse, camera
  - Storage devices, network, printers, display

Hardware and the operating system



Devices alert the CPU that they have a request (keyboard input, disk read return) via an ***interrupt*** (signal sent on the bus). This interrupts what the CPU is doing so that it can respond to (execute code for) the request (interrupt handler).

Brief CPU Processing Overview

- Program instructions and data are in memory
  - Program Counter (PC) register in CPU keeps track of the current instruction location
- CPU stores the next few instructions in cache
  - Some needed data is also cached
  - Multiple layers of cache can be employed
- CPU components typically share the same data width (number of bits)
  - Registers, Arithmetic logic unit (ALU), buses (wires)

Processing Continued

- The CPU is *dumb*
  - It simply continues executing the next instruction until interrupted
    - Fetch -> decode -> execute (repeat)
  - Can only really perform basic arithmetic
- Question:
  - How can we manage these operations and resources effectively?

Answer

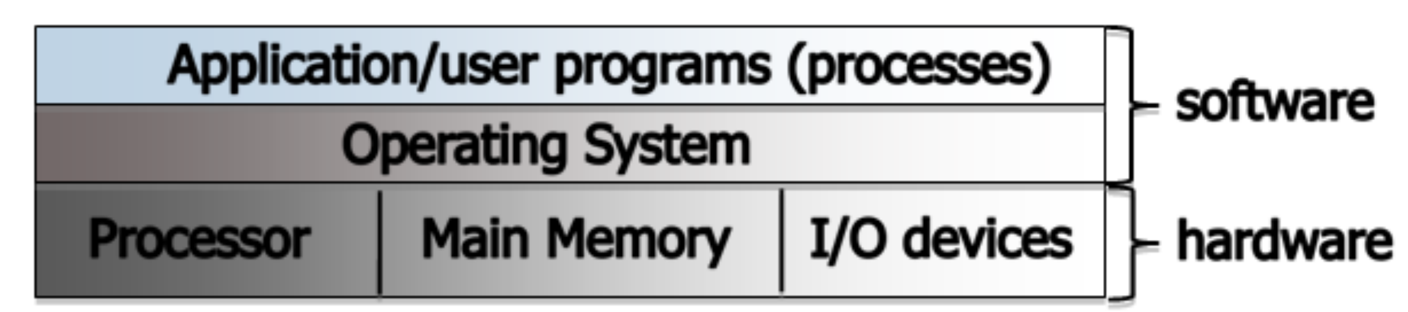
- By using operating system!

Purpose of the Operating System

- Facilitate launching applications
- Manage system resources
- Provide security
- Provide inter-process communication (IPC)
- Additionally OS may:
  - Provide developer libraries
  - Provide program generation tools
    - Text editors, compilers

Two ways to consider the OS

- Bottom-up view
  - OS is software that allocates and de-allocates computer resources – efficiently, fairly, orderly and securely
- Top-down view
  - OS is software that isolates us from the complications of hardware resources
  - In other words, an OS is an application programmer' s and a user's interface to computer operations



Types of Operating Systems

- Single User, Single Process
  - Dos, Windows 3.1
- Single User, Multiprocess
  - Windows 95/98/XP
- Multiuser, Multiprocess
  - Linux, OS X, Windows Server
  - Requires fairness and and security considerations

Consider device latencies/access times

- (all times approximate)
- CPU: 3 cycles per ns
- L1 Cache: 1 ns (3 CPU cycles)
- L2 Cache: 4 ns (12 CPU cycles)
- RAM: 80 ns (240 CPU cycles)
- SSD: 0.1 ms (300,000 CPU cycles)
- HDD: 5 ms (15,000,000 CPU cycles)

Running multiple processes

- Multiprogramming
  - The *yielding* of the CPU to another process when performing IO
- Multitasking (aka timesharing)
  - The forced *yielding* of processes at small intervals to give the impression of concurrently running processes

Multiprocessing benefits

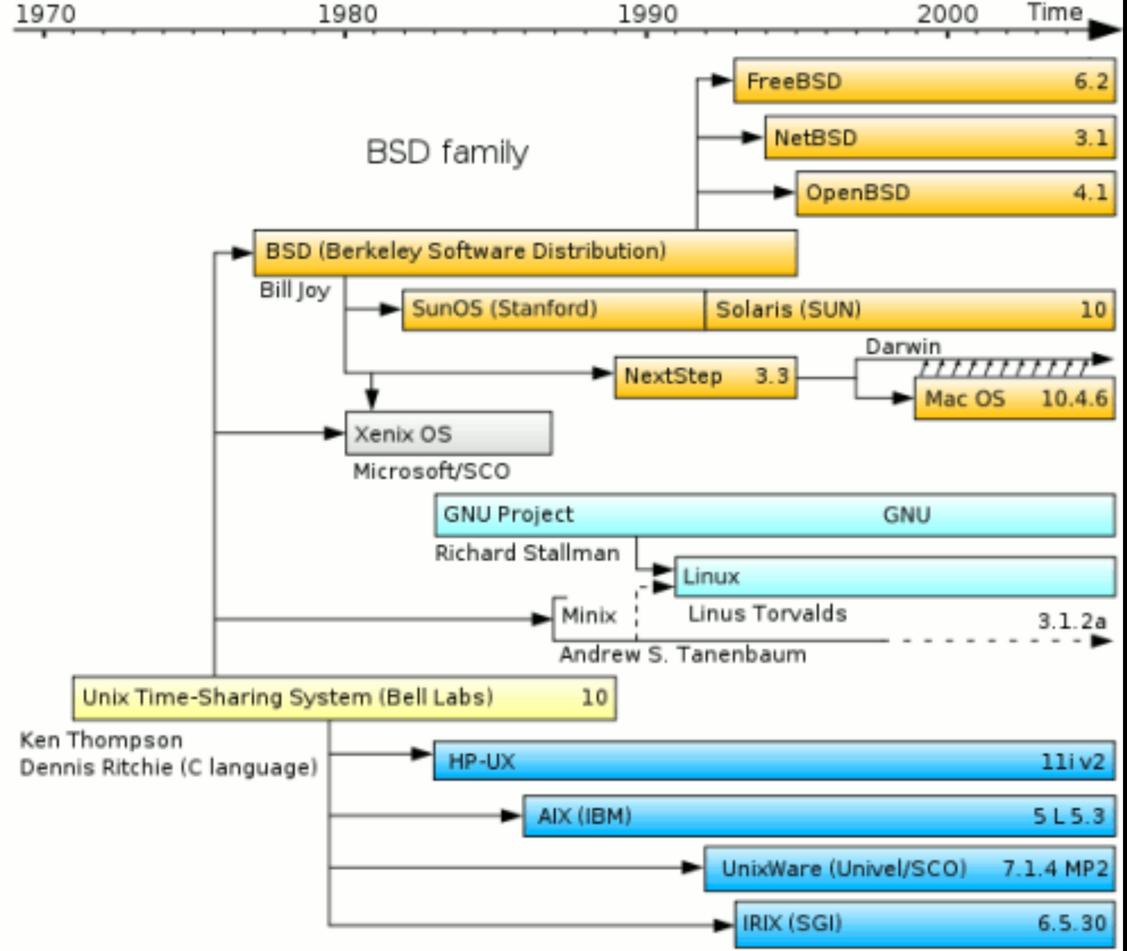
- Increase CPU throughput
  - Perform other operations while waiting on I/O
- Increase resource utilization
  - Resources can maintain a queue of tasks so they always have work to complete

Linux Software Architecture

Brief Unix History

- AT&T Bell Labs – System V standard
  - 1969-70: Ken Thompson wrote Unix in "B"
  - 1972: Dennis Ritchie developed C – a better B
  - Unix rewritten in C, 1973
  - … eventually System V, 1983
- UC Berkeley – BSD standard
  - Started with a copy of System IV, late 1970s
  - Lots of changes/additions in 1980s
  - Now FreeBSD
- Open source - Linux, since early 1990s

Unix-born operating systems



Source: The Abraham Zelmanov Journal
http://zelmanov.ptep-online.com/
linux.html

Unix Philosophy

- Small is beautiful
  - Each program does just one thing
  - Pipe commands (or use successive functions in C) to accomplish more complicated things
  - Less typing is best (using 1970s computers)
    - That's why so many commands are short (ls, cp, mv, …)
- Users/programmers know what they are doing
  - That's what makes the brevity sufficient
  - Means very few restrictions (or safety nets) apply

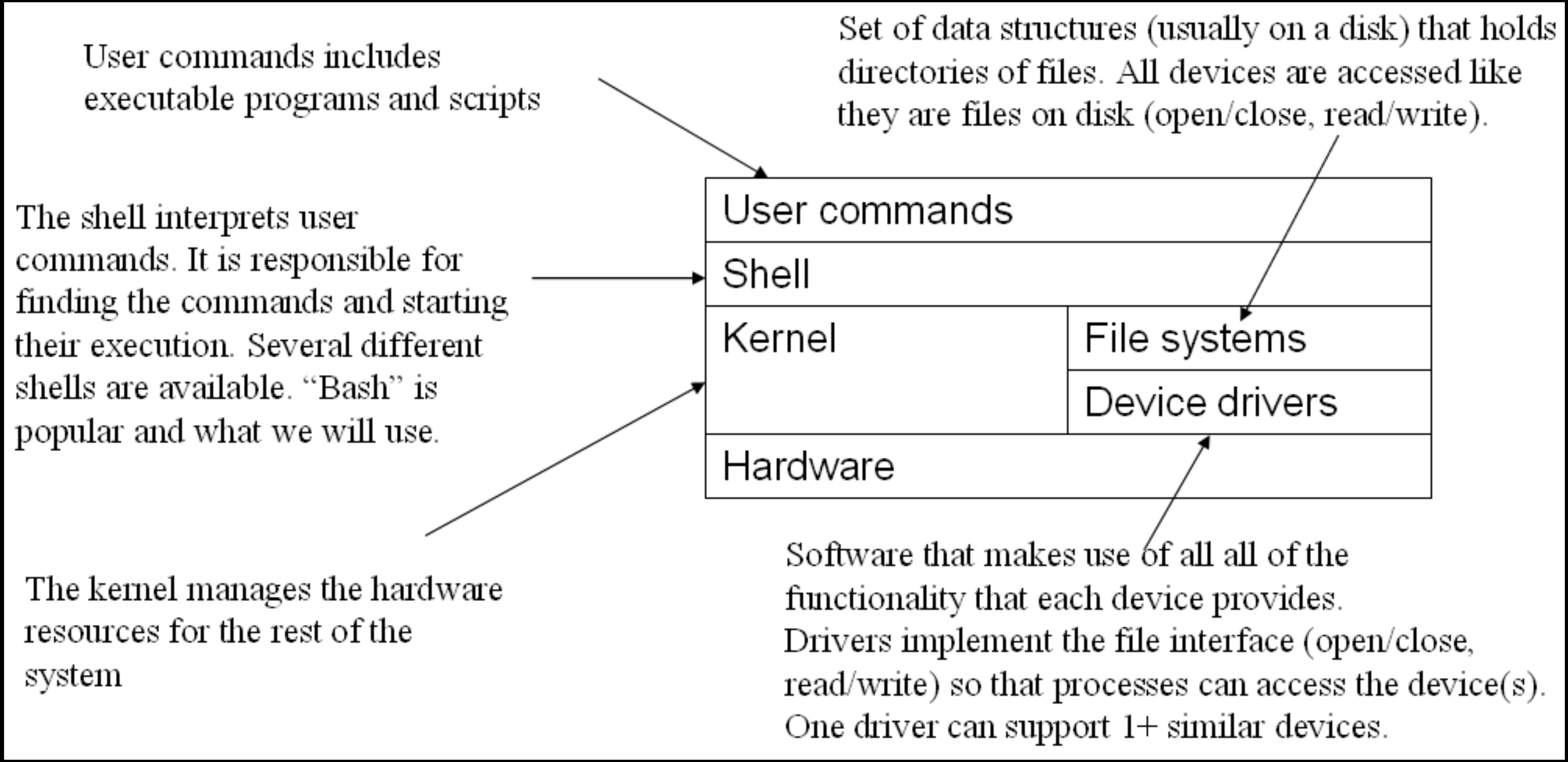
Linux

- Started in 1991 by Linus Torvalds
- Open Source, GPL
  - Free to use, modify, distribute
  - Theoretically allows bugs and security holes to be found faster
- Multi-user, Multitasking OS
- Support for both command line and graphical user interfaces

Linux Distributions

- A Linux distribution is a collection of user-level applications and libraries built around the Linux kernel
- Well known distributions:
  - Ubuntu/Debian
  - CentOS/Fedora/RedHat

Linux Architecture



User commands includes executable programs and scripts

Set of data structures (usually on a disk) that holds directories of files. All devices are accessed like they are files on disk (open/close, read/write).

User commands

Shell

Kernel

File systems

Device drivers

Hardware

The shell interprets user commands. It is responsible for finding the commands and starting their execution. Several different shells are available. "Bash" is popular and what we will use.

The kernel manages the hardware resources for the rest of the system

Software that makes use of all all of the functionality that each device provides. Drivers implement the file interface (open/close, read/write) so that processes can access the device(s). One driver can support 1+ similar devices.

Kernel Services

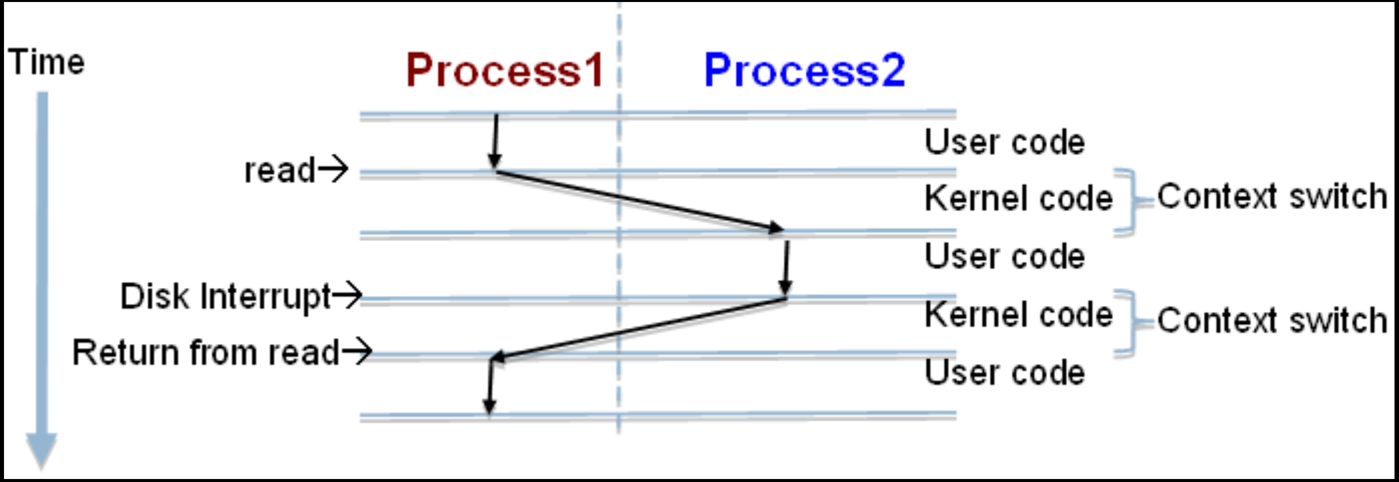
- File management
  - Permissions and access control
  - Manages files and folders
- Process Management and IPC
  - Process scheduling
    - Starting, stopping, suspending, swapping
  - IPC: pipes, named pipes, sockets

Kernel Services cont.

- Memory Management
  - Address spaces for processes
    - Provides isolation between processes and the kernel (hopefully)
  - Manages allocation and de-allocation of memory to processes
- Disk scheduling
  - Mange how processes be given priority to access the disk?

CPU Scheduling

- Kernel sends interrupt to a process to give another process a turn to use the CPU
- Processes can give up CPU when they don't need it (e.g. waiting on I/O device)



Processes request kernel services

- Using system calls (read, write, fork, …)
  - – OOP idea: these are the kernel’s interface
  - – Processes access devices just like files – that’s how they are represented by the kernel, and they occupy places in the file system
    - Use open, close, read, write, release, seek, …
- Or indirectly, by using shell commands or libraries/programs that use system calls

A few system calls

- open: open a “file”
- read: read data from a “file”
- write: write data to a “file”
- exec: begin executing a new program
- fork: start a new process as a copy of the current one

Example Library Function Chain

- fclose: posix C close file stream function <stdio.h>
- close: unix close file descriptor function <unistd.h>
- Invokes the following system call (assembly)

```
mov ebx, 0  # indicate we want to close fd 0
mov eax, 6  # system call number 6 is close
int 80h  # send interrupt 80 for system calls
```

For tomorrow

- Get the reader if you don’t already have it
- Finish the first section of the reader “Introduction to operating systems, Unix and shells.”
- Begin reading section 3 of the reader “Processes”