EC 440 – Introduction to Operating Systems

Manuel Egele

Department of Electrical & Computer Engineering

Boston University

EC 440 – Course Staff

Manuel Egele

Office Hours : Mon. & Wed. 18:15—19:00 (conceptual Qs only)
Location: After class or PHO337

Sadie Allen (GTF)

OHs: Tue. & Thu. 18:30 – 20:30
Location: PHO305

EC 440 Info

Main resource: Piazza

https://piazza.com/bu/fall2022/ec440

- 1st stop for questions (check for duplicates!)
- Help each other with answers (**not solutions!**)
- Resources (lecture slides, etc.)
- Projects/homeworks/challenges will be posted there too

Requirements

The course requirements include

- several projects (homework)
- a midterm and a final exam

The projects (and exams) are *individual* efforts

Final grade will be determined as follows:

0 on quiz ->
0 on homework

- projects: 60% (5x (12% auto-grader x oral quiz))
- exams: 35% (Mid-term: probably Oct. 19th (15%), Final: TBD (20%))
- participation: 5% (incl. Piazza, In-class Quizzes, Handouts)
- no, there won't be a curve; < 50%, (F)ail; >= 50% equally distributed

Academic Integrity

- Projects (& exams) are <u>*individual*</u> efforts
- Feel free to <u>*discuss*</u> problems and their solutions with your class-mates
- Do not, under <u>any circumstances</u> share or copy any amount of code
  - "Oh I wanted to see whether my friend's code passes the tests when I submit it."
  - "Oh I found it on stack overflow and it just seems to work."
- Violators will be identified and reported to the academic misconduct panel
- Follow BU's academic Conduct Code

  https://bu.edu/academics/policies/academic-conduct-code/

Grading Scale

- 100% is full score, each 5.55% is one step

A	> 94.44%
A-	> 88.88%
B+	> 83.33%
B	> 77.77%
B-	> 72.22%
C+	> 66.66%
C	> 61.11%
C-	> 55.55%
D	> 50%
F	<= 50%

Projects

~5 programming assignments

1) **Shell (system calls)**
2) **Threads (parallel execution, scheduling)**
3) **Synchronization (semaphores, …)**
4) **Memory (virtual memory, shared regions, …)**
5) **File systems**

No late submission!

Individual effort! (i.e., NO duplicate code-***snippets***)

Academic conduct will be ***strictly*** enforced.

https://www.bu.edu/academics/policies/academic-conduct-code

i.e., you can discuss problems and potential solutions with others, ***but*** you must not write code together (or use others' code)!

Expectations (Of You)

- Technical interest in learning about OS
  - Taking this course w/o interest is useless
- Knowledge in
  - Programming (Especially: C, gdb)
  - Computer Architecture (Especially: Stack)
- Patience
  - The programming assignments are ***really*** hard & rewarding (most take me ~ 8hrs each, 2016 ~20hrs each self-reported by students)
- Adhere to and uphold course policy
  - **No copying** (challenges are individual effort!)
  - Stay **ethical**
  - **Cite any & all resources you use** (citation is not an excuse for copying)
  - Completely **understand** any & **all code you submit**

Expectations (Of Me)

- I will try to answer all your questions
  - I do not have all the answers, but I should be able to give useful pointers for most things
- I will make the course as practical as possible
  - i.e., you will get the chance to gain a lot of hands-on experience
- I will try to cover a good mix between foundational and practically applicable topics in this course

Material

Slides will be posted on Piazza after each lecture

The course will not adopt any books, but:

Two classic OS Books:

Andrew S. Tanenbaum and Albert S. Woodhull
Operating Systems (Design and Implementation)
3rd Edition, Prentice-Hall, 2006

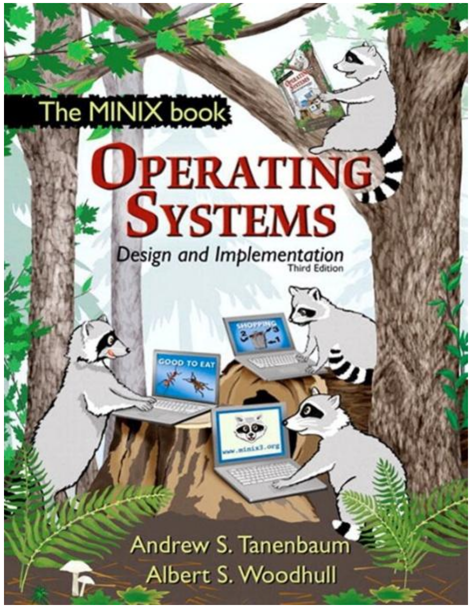
Andrew S. Tanenbaum and Herbert Bos
Modern Operating Systems
Prentice-Hall, 2015

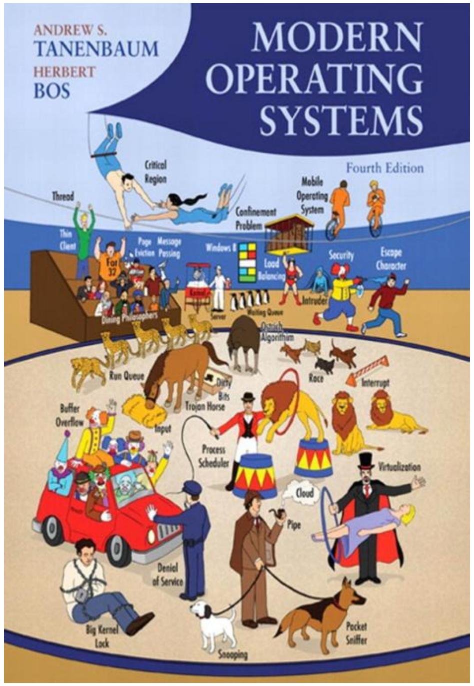
Also important (if you need to brush up your C):

Brian W. Kernighan and Dennis M. Ritchie

The C Programming Language
2^{nd} Edition Prentice-Hall, 1988

Excellent resources, but not strictly mandatory/required.





Operating Systems

Let us do amazing things …

- allow you to run multiple programs at the same time
- protect all other programs when one app crashes
- allow programs to use more memory than your computer has RAM
- allow you to plug in a device and just use it (well, most of the time)
- protects your data from fellow students on eng-grid

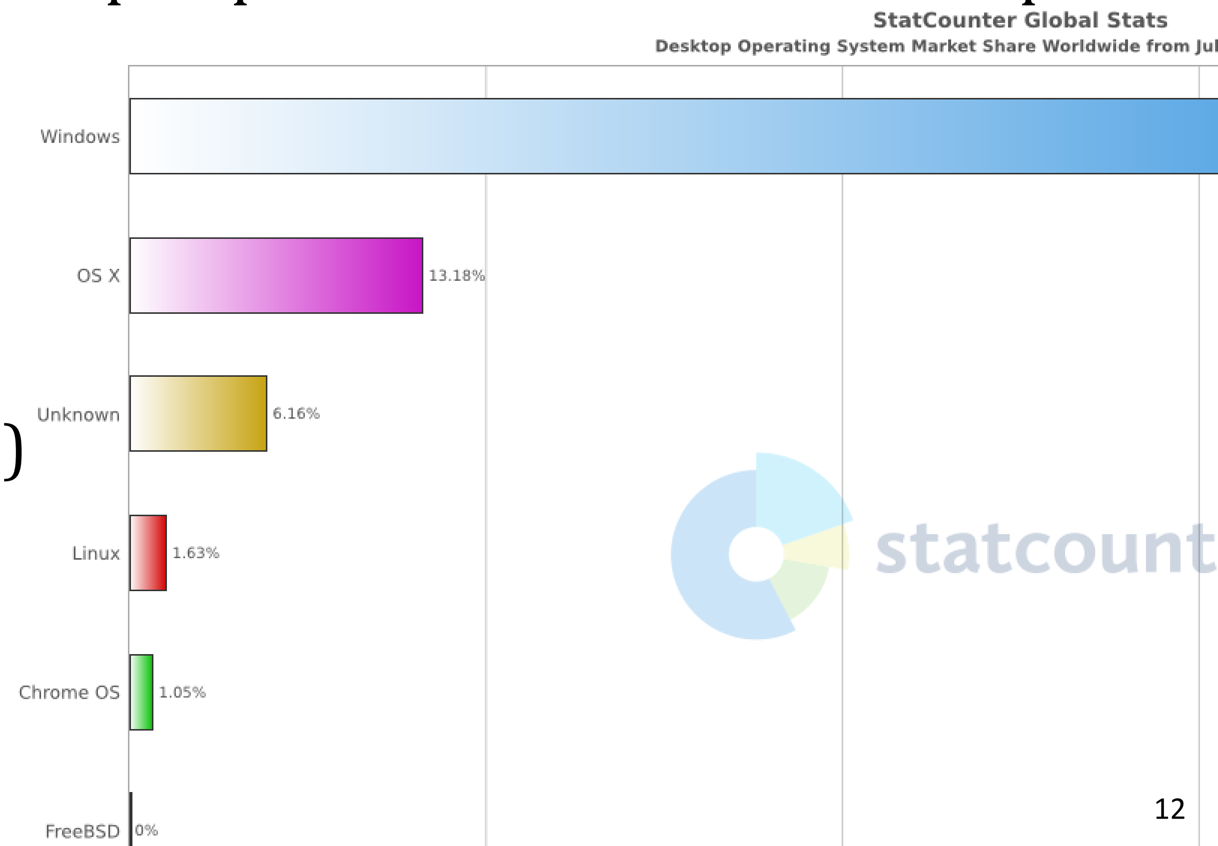
What is the Most-Used OS?

Desktop Operating Systems

– Microsoft Windows
  - sells millions of copies per month and ~78% desktop market share

– Apple OS X
  - ~13% share

– Linux
  - negligible (1%)



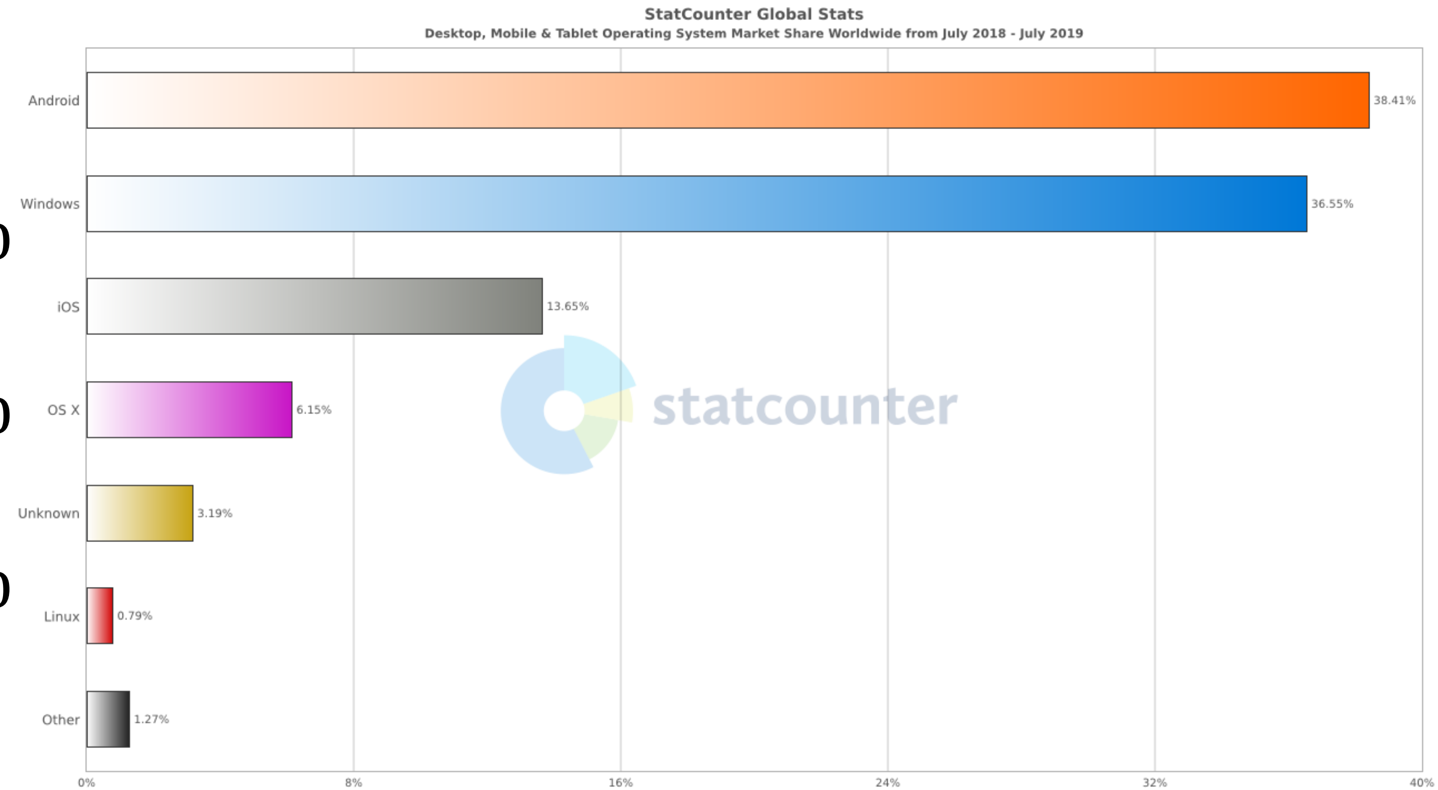
What is the Most-Used OS?

Computers that browse the Web

~36%

~38%

~19%



What is the Most-Used OS?

But wait … what about embedded devices?

– order of magnitude more devices

iTron (several billion installations)

Wind River (VxWorks) – market leader, “Lord of Toasters”

Linux is growing rapidly

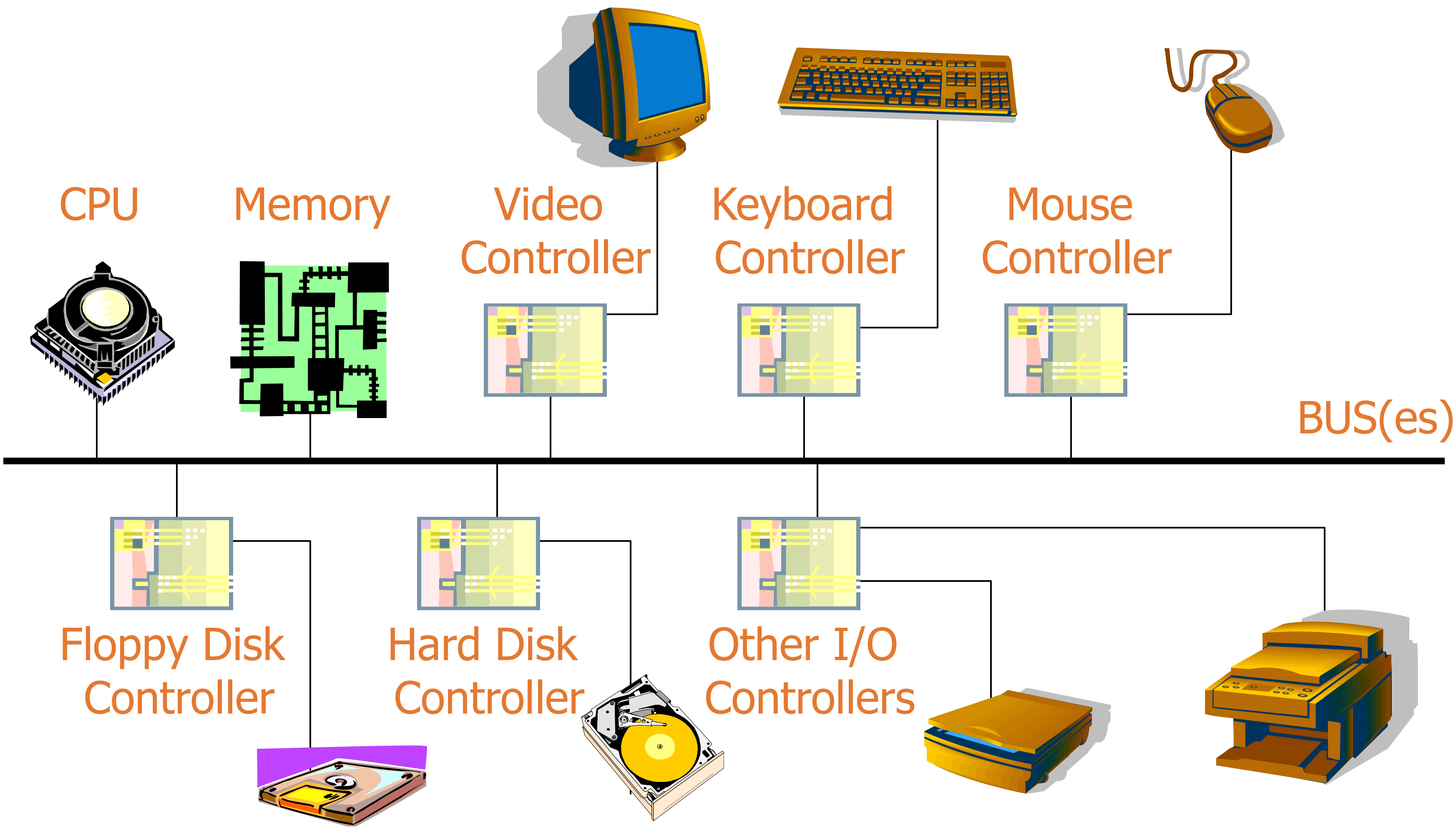
Minix in Every Modern Intel System

- Intel Management Engine (part of *every* Intel system since ~2008) runs a version of Minix

Outline

1) Introduction to Operating Systems
2) Processes and Threads
3) IPC, Synchronization, and Deadlocks
4) Memory Management
5) Input/Output
6) File Systems
7) Security

In The Beginning There Was Hardware



Central Processing Unit

Fetches instructions from memory and executes them

Characterized by an *instruction set*

- Loads and stores values to/from memory/registers
- Performs simple operations (add, and, xor)
- Jumps to locations

Contains a set of *registers*

- Program counter
- Stack pointer
- PSW (Program Status Word)
  - Kernel mode: total access to memory/registers and instructions
  - User mode: limited access to memory/registers and subset of instructions

Memory

Set of locations that can hold values

Organized in a hierarchy of layers

- Registers (access time ~1 nsec)
- Cache memory (access time ~2 nsec)
- Main memory - RAM (access time ~10 nsec)
- Hard disk (access time ~10 msec)

Read Only Memory (ROM) used to store values … permanently

I/O Devices

Controllers connected to the bus

Device connected to a controller

The controller provides an interface to access the device resources/functionalities

- done by storing values into device registers

Memory mapped access

- device registers mapped into memory region

Dedicated I/O

- special CPU instructions

Disk

One or more metal platters that rotate at a constant speed (e.g., 5400 rpm, 7200 rpm, …)

Each platter has many concentric *tracks*

Corresponding tracks in different platters compose a *cylinder*

Each track is divided in *sectors*

A mechanical arm with a set of heads (one per surface) moves on platters and reads/writes sectors

- Move to the right track (1 to 10 msec)
- Wait for the sector to appear under the head (5 to 10 msec)
- Perform the actual read/write

And, of course, there are solid state drives (SSDs)

Buses

Used to transfer data among components
Different functions, speeds, number of bytes transferred

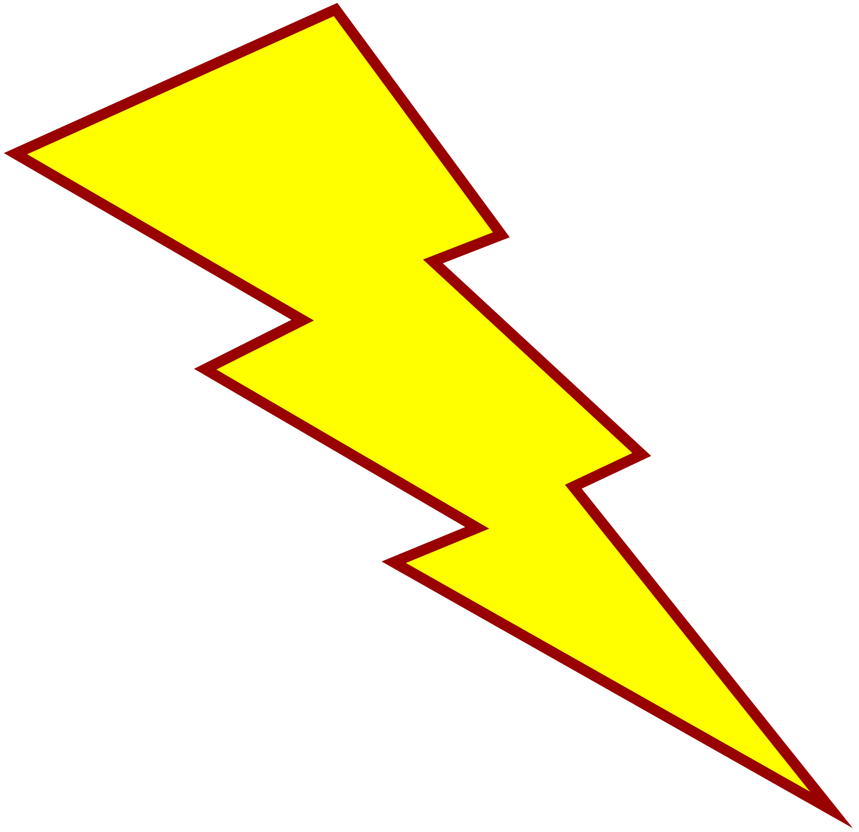
Cache bus
Memory bus (FrontSide Bus, QuickPath Interconnect)

ISA (Industry Standard Architecture) bus
- 8.33 MHz, 2 bytes, 16.67 MB/sec

PCI (Peripheral Component Interconnect) bus
- 66 MHz, 8 bytes, 528 MB/sec

PCIe (PCI Express)
USB (Universal Serial Bus)
SCSI (Small Computer System Interface) bus
IEEE 1394/FireWire bus

There Be Power…



There Be Power…

CPU starts and loads instructions starting at 0xfffffff0

Instruction jumps to BIOS code

BIOS (Basic Input/Output System) is started

- Performs basic tests (memory, keyboard, etc) – POST (power on self test)
- Determines the "boot device" (Hard disk, Floppy, CD-ROM)
- Loads the contents of the first physical sector (the Master Boot Record - MBR - Cyl 0, Head 0, Sect 1) in memory 0x7C00 - 0x7DFF
- Jumps to 0x7C00

MBR code finds an "active" file system, loads the corresponding boot sector in memory, and jumps to it

The boot sector code loads the *operating system*

A few words about the BIOS

Two main components

- Boot services
  - initialize hardware (including RAM)
  - read and load boot code
  - transfer control
- Runtime services
  - basic routines for accessing devices
  - can display menus, boot from devices (and even network)
  - access to clock, NVRAM, …
  - OS typically bring their own device drivers

Developments

- Standard PC BIOS around for a long time (~1975)
- recently, Unified Extensible Firmware Interface (UEFI) started as replacement
- UEFI is more general, supports boot from large disks

The Operating System

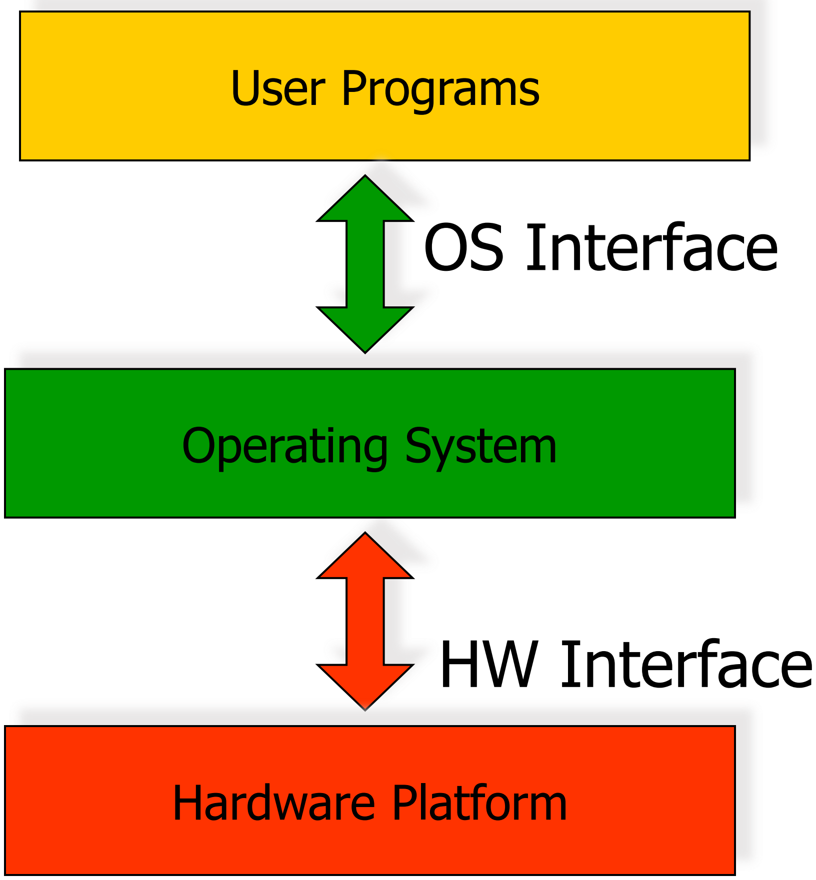
Where does an operating system fit in a computing system?

What does an operating system do?

Why do we need an operating system?

How is an operating system structured?

Where?


User Programs
OS Interface
Operating System
HW Interface
Hardware Platform

What?

The operating system is a *resource manager* that provides an orderly and controlled allocation of resources

The operating system is an implementer of *multiple virtual (extended) machines* that are *easier to program* than the underlying hardware

Goal:

- Each program/application can be developed as if the whole computer were dedicated to its execution

Resource Management

Multiplexing

- creating an illusion of multiple (logical) resources from a single (physical) one

Allocation

- keep track of who has the right to use what

Transformation

- creating a new resource (logical) from existing resource (physical) primarily for "ease of use"

An OS multiplexes, allocates, and transforms HW resources

Types of Multiplexing

Time multiplexing

- time-sharing
- scheduling a serially-reusable resource among several users
  - e.g., CPU or printer

Space multiplexing

- space-sharing
- dividing a multiple-use resource up among several users
  - e.g., memory, disk space

Multiple Virtual Computers

Multiple processors

- capability to execute multiple flows of instructions at the same time

Multiple memories

- capability to store information of multiple applications at the same time

Access to file system as an abstraction of the disk

Access to other I/O devices in abstract, uniform ways

- e.g., as objects or files

Virtual Computers

OS creates multiple processes (simulated processors) out of the single CPU

- time-multiplexing the CPU

OS creates multiple address spaces (memory for a process to execute in) out of the physical memory (RAM)

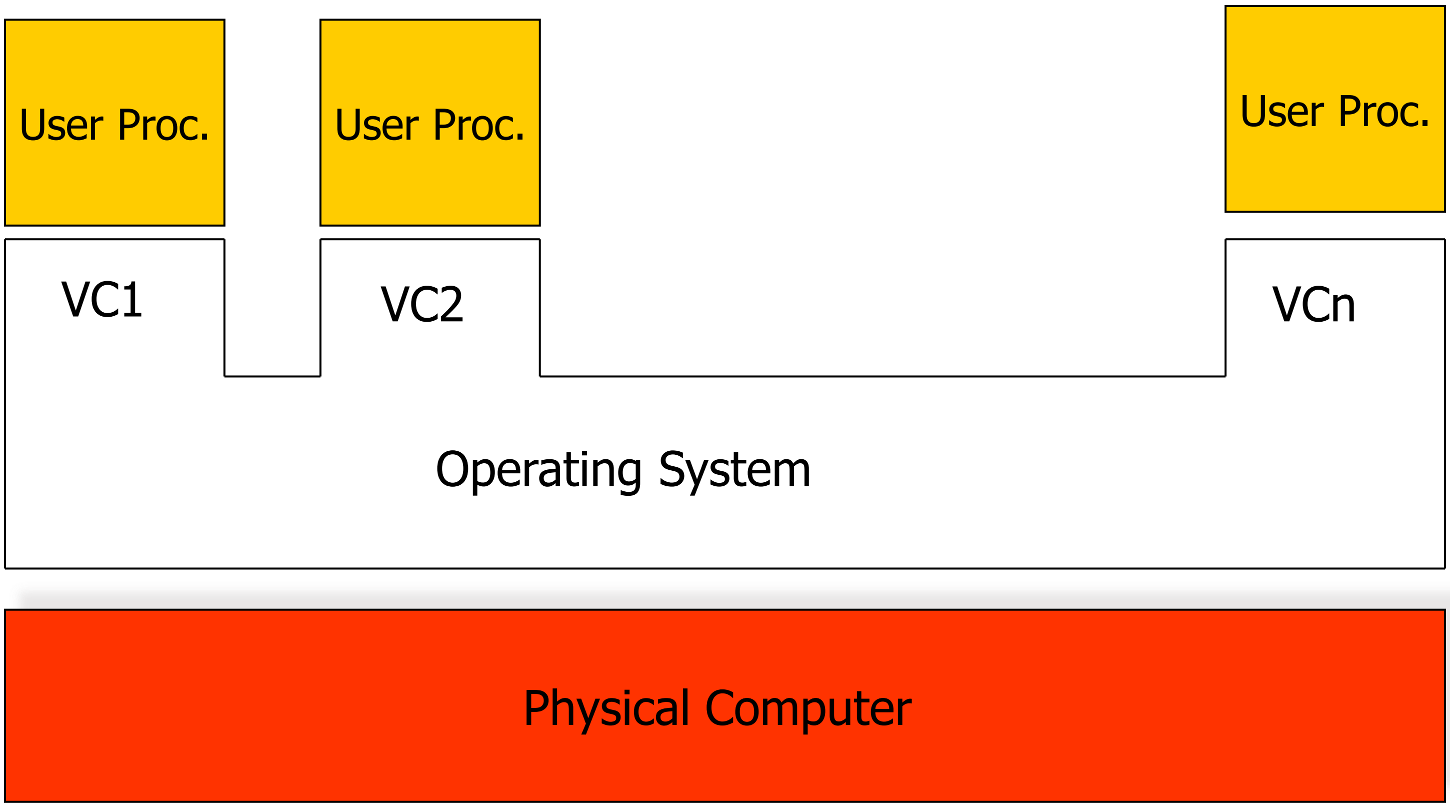
- space-multiplexing of the memory

OS implements a file-system and I/O system so that processes use and share the disks and I/O simultaneously

- space-multiplexing the disk and time-multiplexing the I/O channels

OS creates multiple virtual computers from a single physical machine

Virtual Computers



OS Interface – Virtual Processors

Nearly the same interface as the physical CPU

OS removes privileged operations

- PSW determines if the code is either “user code” or “OS code”
- changes in status are strictly regulated...

OS adds instructions (system calls)

- create new virtual computers (processes)
- communicate with other VCs
- allocate memory
- perform input and output operations I/O
- access the file system

OS Interface – Virtual Memory

- **Memory of the Virtual Computer is similar to the hardware memory (i.e., a sequence of words), and it is accessed the same way**
- **The OS divides up the memory into parts and gives each part to each virtual computer**
- **OS creates an illusion that each virtual computer has a memory starting from address 0x0000**
- **OS creates an illusion that the virtual computer has more memory than the physical memory**

OS Interface – Virtual File System

- **Secondary storage provides long-term storage for data**
- **Storage is done physically in term of disk sectors and virtually in terms of disk files**
- **The virtual computer sees a file system consisting of named files with arbitrary size**

OS Interface – Virtual I/O

- **I/O operations of the virtual computer are completely different from the I/O operations of the physical computer**
- **The physical computer has devices with complex control and status registers**
- **In contrast, the virtual I/O is simple and easy to use**
- **In fact, in most OSes (e.g., UNIX) virtual I/O abstraction is almost identical to the file-system interface giving rise to uniformity of treatment with respect to all types of I/O devices including disks, terminals, printers, network connections**
- **Each VC sees a dedicated I/O device: the actual hardware is space/time multiplexed by the OS**

Operating System Services

The programs running on virtual computers access the operating system services through *system calls*

A system call is carried out by

- Storing the *parameters* of the system calls in specific locations (registers, memory)
- Calling a "software interrupt" or "trap"

Switch to kernel mode: the OS is notified and takes control of the situation

Do We Need an OS?

- **For a specialized application (e.g., a microwave oven), an OS may not be needed**
- **The hardware can directly be programmed with the rudimentary functionalities required by these applications**
- **A general-purpose computer, on the other hand, needs to run a wide range of user programs**
- **For such a system, an OS is indeed necessary**
- **Otherwise, each user will need to program their own operating system services**
- **An OS can do this for once and make it available to the user programs**

Self Assessment

Today go to: https://bit.ly/3lD6y6I

If you have trouble with the C program (Q10) you'll definitely want to brush up on your C-skills.