ECS518U - Operating Systems
Week 1

Introduction:
Operating System Concepts

Tassos Tombros

Learning Objectives

By the end of the lecture you should be able to:

- Describe **what an OS is**
  - What are the main functions of an OS
  - Some historical perspectives
- Discuss some **main concepts central to OSs (preview / pointers to the weeks to come)**
  - Kernel & dual mode operation
  - Processes
  - Memory
  - Files and I/O
- Identify the main **interfaces to the OS**
  - Shell(s)
  - System calls

Outline

- **What is an OS?**
  - What an OS does
  - (a little bit of) Historical perspective
- **Main concepts**
  - Processes
  - Memory
  - Files and I/O
- **Interfaces to the OS**
  - System calls, shell(s)
- Information about module organisation & **assessment**
- Resources for Exploring OS

Note: some material are from Tanebaum's book

Why learn about Operating Systems?

- An OS is the **largest and most complicated software** running on most machines
- By knowing how operating systems work, we learn:
  - how to organise a large piece of s/w (**design**)
  - how to hide complexity (**abstraction**)
  - how to tune a large system (**performance**)
  - how to share s/w and h/w components in a safe, efficient and fair way (**resource management**)
- **We also become better programmers**
  - when we write a program, this runs on an OS running on some machine (a programmer is a user of an OS)
  - understanding how OSs work helps you optimise your code and make better use of the OS

What is an Operating System?

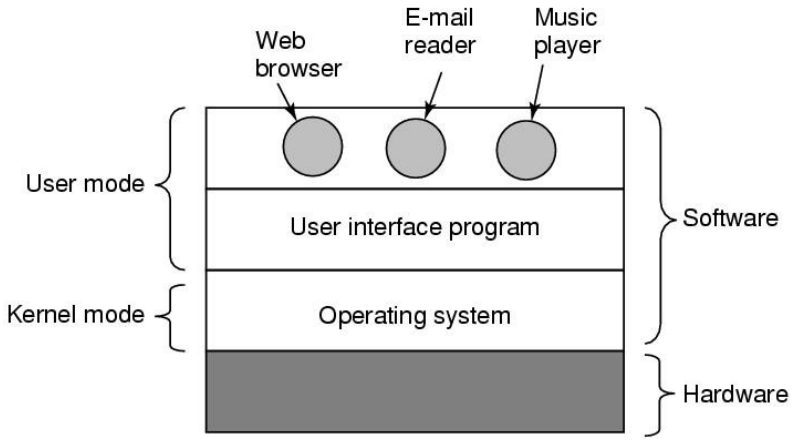
- **An OS creates a computer that is convenient and easy to use from the underlying hardware**
  - today's computers are so easy to use that we do not notice how this is achieved
  - in some sense, the OS is a solution to a s/w engineering problem: **how do you hide the complexity and limitations of hardware from application programmers?**
- We can see the OS as **software** that **manages the computer's resources (in a fair and secure way)** for its users and applications
  - makes it easier to write applications
  - makes it easier to run applications efficiently

Hardware Interface

- **OS is between the applications and the h/w**
  - h/w is 'ugly' – complex
- OS provides us with a standard and more **abstract way to communicate with the h/w**

Main functions:

- Memory management
- CPU scheduling, process management
- I/O management
- File and storage management
- Networking
- More? Possibly, no universally agreed 'definition'

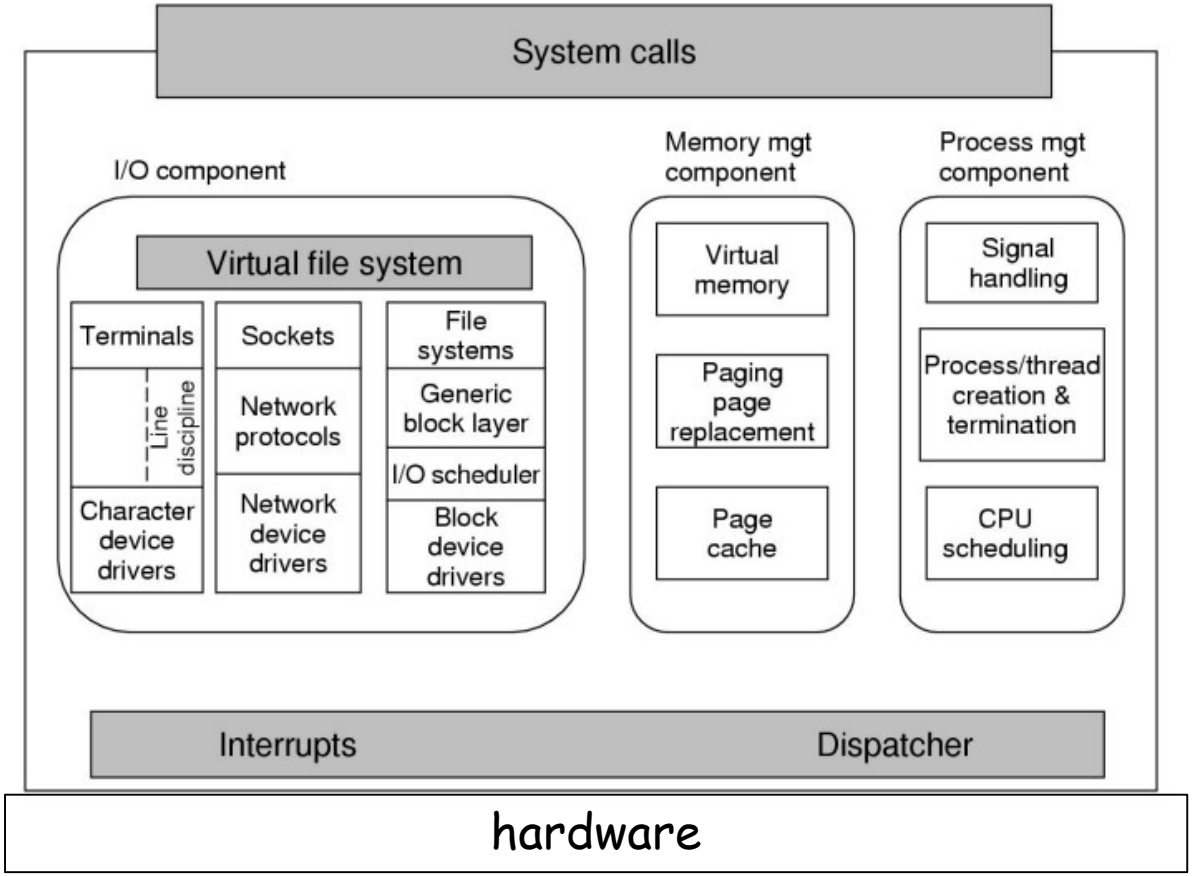


Example Kernel Structure (Linux)

"The one program running at all times on the computer" is the **kernel**.

The kernel is the core of the OS:

- Has complete control over everything that occurs in the system
- Able to call all processor instructions and registers

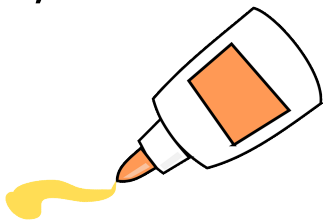


- Kernel operates in **privileged (kernel) mode** whereas all user programs and apps operate in **user mode**
- **Dual mode operation** prevents user data from interfering with kernel data (benefits e.g. for system reliability, security, etc.)

Figure taken from Tanenbaum

Roles of an OS

- **Referee (you can always blame the referee!!!)**
  - **Resource allocation** among users, applications
  - **Isolation** of different users and applications from each other (**protection**)
  - **Communication** between users, applications
- **Illusionist**
  - Each application appears to have the entire machine to itself
  - Infinite number of processors, (near) infinite amount of memory, reliable storage, reliable network transport, etc.
- **Glue**
  - Offers a set of common services
  - Storage, user interface widgets, networking, sharing, …



Roles of an OS – referee example

- **Problem:** Run multiple applications in such a way that they are protected from one another
- **Goal:**
  - Keep user programs from crashing the OS
  - Keep user programs from crashing each other
- (Some of the required) Mechanisms:
  - **Address Translation**
  - **Dual Mode Operation** (user – kernel mode)
- **Simple Policy**:
  - Programs are not allowed to read/write memory of other Programs or of Operating System

(a little bit of) History

- Mainframes
  - Multi-programming
  - Time sharing
- Personal computer

Understanding the Problem

- **It is important to understand the problem each aspect of an OS solves**
  - WHY does it do … ?

- Why multi-tasking?
  - **Multi-tasking improved the efficiency of the use of the CPU** because much of the time the program was not using the full resources of the computer as it was waiting for an I/O device (e.g. printer)

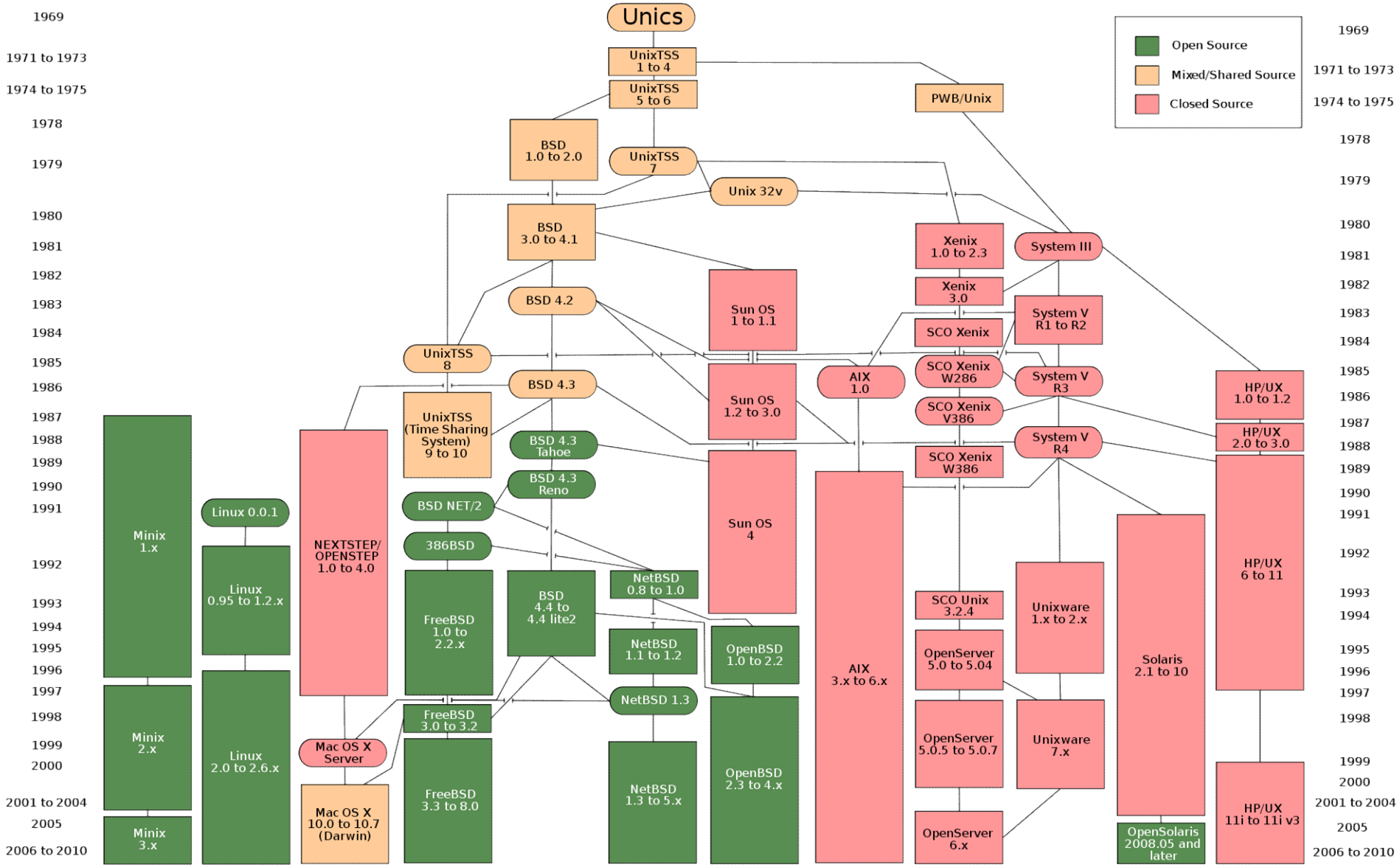
How Much is the OS?

- **Is the GUI part of the OS?**
  - Linux has a choice of window managers
    - KDE, GNOME, etc.
  - Xwindows
- Is the (should the) browser (be) part of the OS?
  - ChromeOS?
  - United States vs. Microsoft (1998-2001)
    - Microsoft accused of monopoly behaviour
    - Can IE be bundled with Windows?
    - MS argued that IE is part of Windows
    - http://en.wikipedia.org/wiki/United_States_v._Microsoft

Variety of Operating Systems

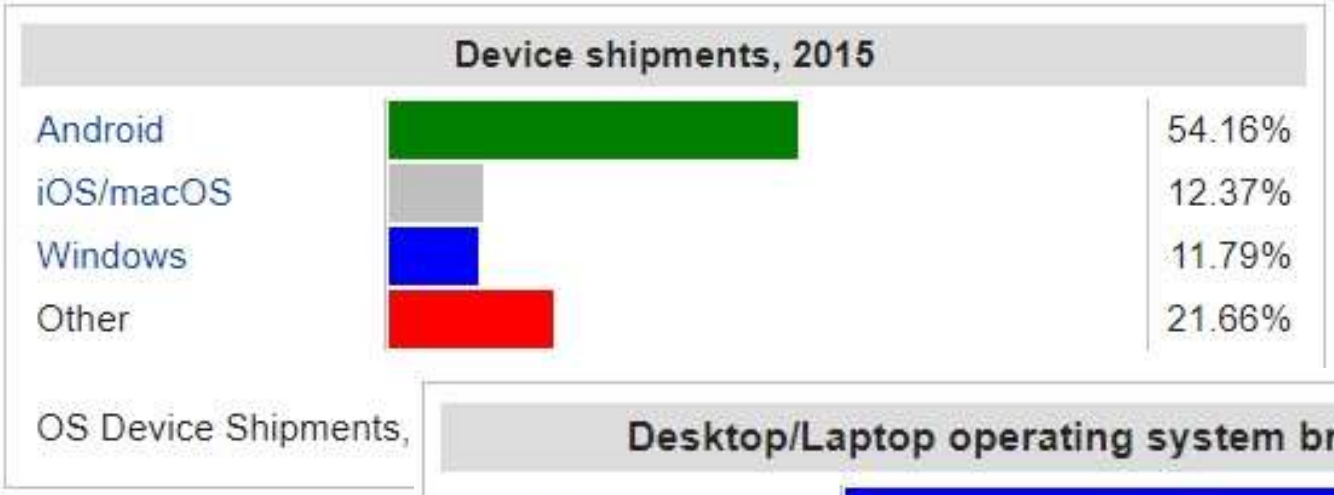
- Mainframes
  - IBM z/OS, also Linux, Solaris
- Servers
  - Linux (2/3) and Windows (1/3)
- **Supercomputers**
- **PCs (desktop, laptop)**
- **Phones and PDAs**
- Real-time embedded systems
- **Wearable devices**, InternetOfThings(IoT)
  - AndroidWear, Tizen, etc.

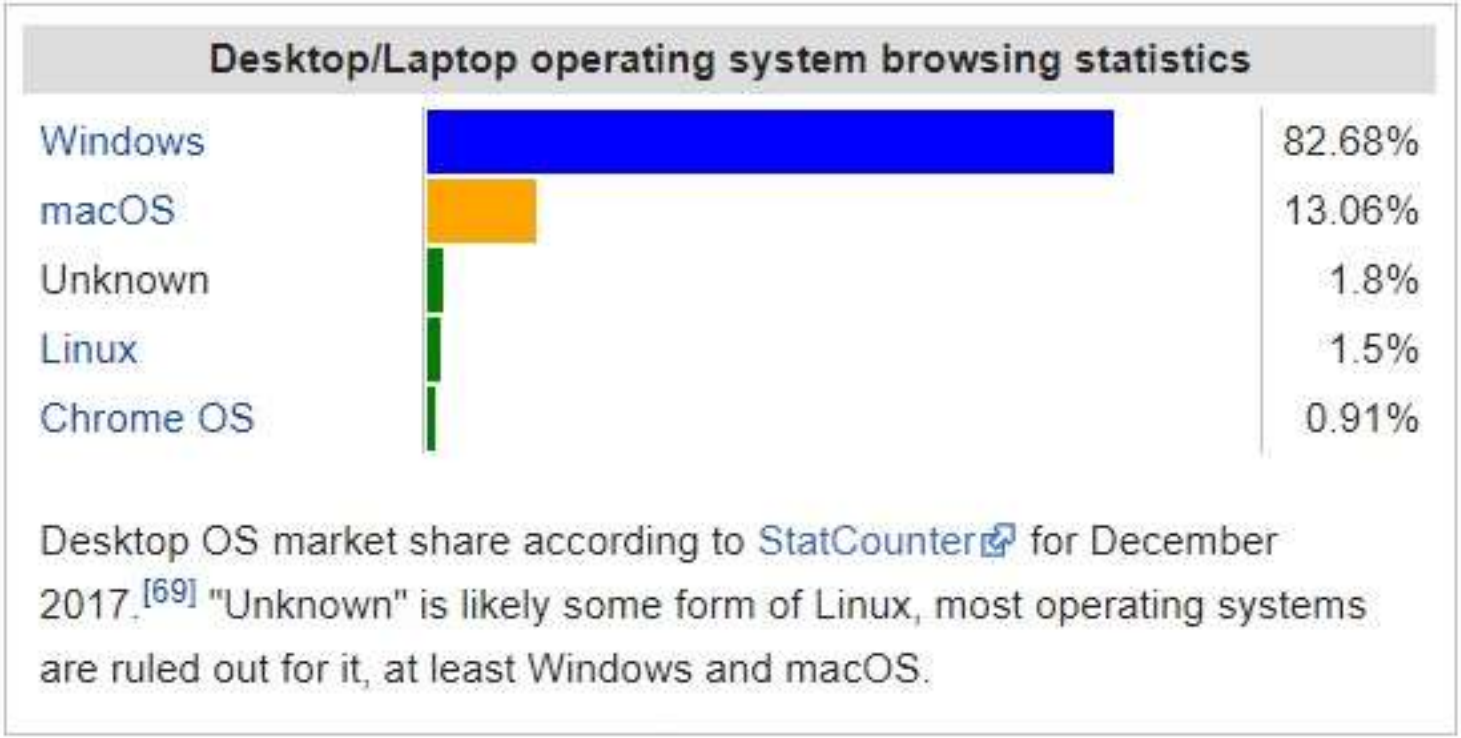
Linux / Unix main example on this module


Unics
UnixTSS 1 to 4
UnixTSS 5 to 6
UnixTSS 7
UnixTSS 8
UnixTSS (Time Sharing System) 9 to 10
PWB/Unix
Unix 32v
BSD 1.0 to 2.0
BSD 3.0 to 4.1
BSD 4.2
BSD 4.3
BSD 4.3 Tahoe
BSD 4.3 Reno
BSD NET/2
386BSD
BSD 4.4 to 4.4 lite2
FreeBSD 1.0 to 2.2.x
FreeBSD 3.0 to 3.2
FreeBSD 3.3 to 8.0
NetBSD 0.8 to 1.0
NetBSD 1.1 to 1.2
NetBSD 1.3
NetBSD 1.3 to 5.x
OpenBSD 1.0 to 2.2
OpenBSD 2.3 to 4.x
Sun OS 1 to 1.1
Sun OS 1.2 to 3.0
Sun OS 4
AIX 1.0
AIX 3.x to 6.x
Xenix 1.0 to 2.3
Xenix 3.0
SCO Xenix
SCO Xenix W286
SCO Xenix V386
SCO Xenix W386
SCO Unix 3.2.4
OpenServer 5.0 to 5.04
OpenServer 5.0.5 to 5.0.7
OpenServer 6.x
System III
System V R1 to R2
System V R3
System V R4
Unixware 1.x to 2.x
Unixware 7.x
Solaris 2.1 to 10
OpenSolaris 2008.05 and later
HP/UX 1.0 to 1.2
HP/UX 2.0 to 3.0
HP/UX 6 to 11
HP/UX 11i to 11i v3
Minix 1.x
Minix 2.x
Minix 3.x
Linux 0.0.1
Linux 0.95 to 1.2.x
Linux 2.0 to 2.6.x
NEXTSTEP/ OPENSTEP 1.0 to 4.0
Mac OS X Server
Mac OS X 10.0 to 10.7 (Darwin)
Open Source
Mixed/Shared Source
Closed Source
1969
1971 to 1973
1974 to 1975
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001 to 2004
2005
2006 to 2010

OS Share: Web Clients

- From http://en.wikipedia.org/wiki/Usage_share_of_operating_systems





Technology & Market Evolution

	1980	1997	2014	Factor (1980-2014)
Single CPU Speed (MIPS)	1	1000	2500	2500
CPUs per computer	1	1	10+	10+
Cost of MIPS/$	$100K	$25	$0.20	500K
Memory capacity (MB/$)	0.002	2	1K	500K
Disk capacity (GB/$)	0.003	7	25K	1
Ratio of users to computer	100:1	1:1	1: several	100+

Smartphone shipments exceed PC shipments

2011 shipments:

- 487M smartphones
- 414M PC clients
  - 210M notebooks
  - 112M desktops
  - 63M tablets
- 25M smart TVs

341.6M in Q2 2017

61.1M in Q2 2017

37.9M in Q2 2017

over 100M in 2015

The end of Moore's law

- At the age of 51 **Moore's law is officially dead**
  - "The number of transistors per chip would double every 12 months." -> adjusted in 1975 to "roughly every 24 months"

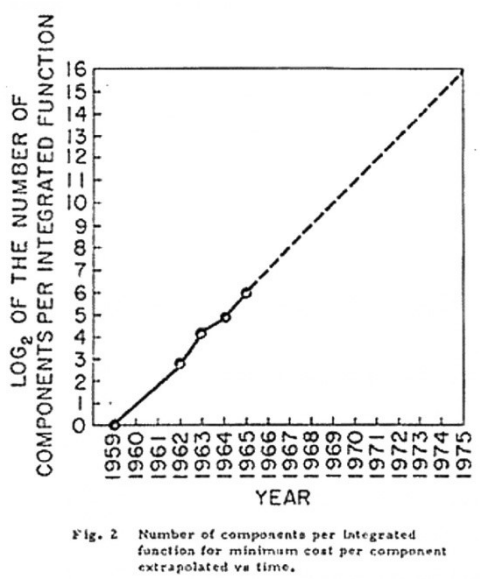


Fig. 2 Number of components per integrated function for minimum cost per component extrapolated vs time.

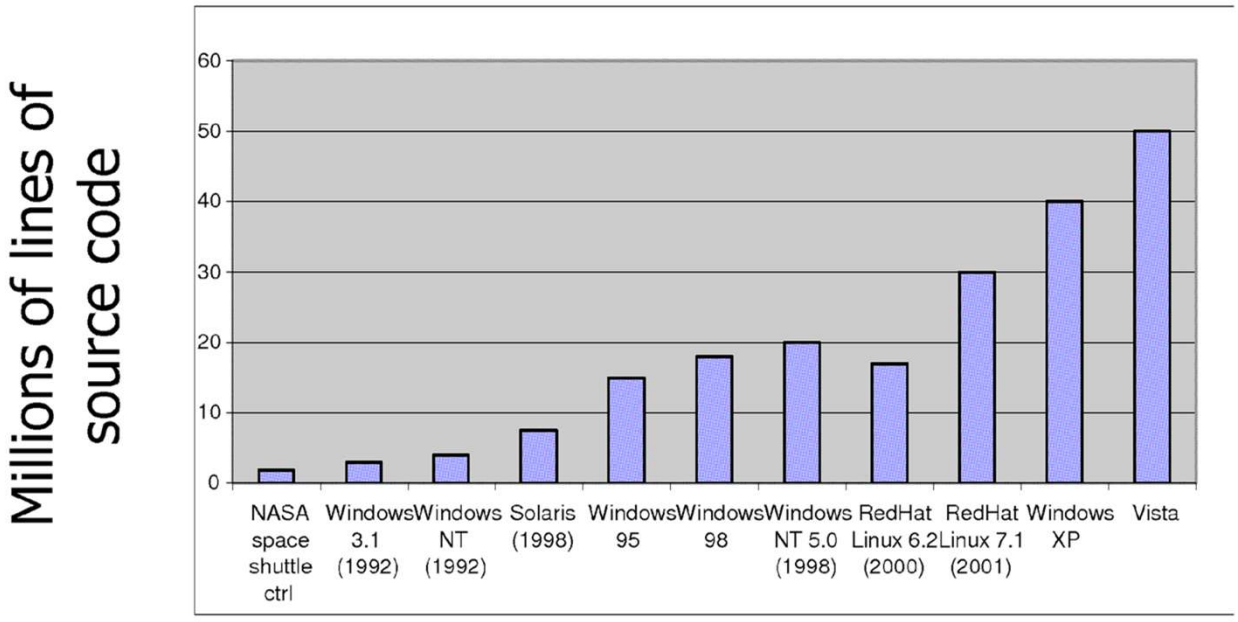
Why should we care?

- Chip fabrication has hit physical limitations
- **Big part of the future focus will be on IoT, connected devices (sensors & low power processors & applications/OSs for such uses)**

Source: http://arstechnica.co.uk/information-technology/2016/02/moores-law-really-is-dead-this-time/

Trivia

- What is the **main cause of OS crashes (≈50%)?**
  - "Bad" device drivers – usually 3rd party drivers
- What **languages** are OSs (mostly) written in?
  - C, C++ and Assembly for lots of low-level operations
- **Lines of code** involved in OSs (how many?)



From MIT's 6.033 course

CHECK POINT 1

- What is an OS?

- List the main functions of an OS

- Explain how you achieve protection with dual mode operation

- **The OS as an illusionist – what are some illusions offered?** (**2017 May paper, Q1a:** One of the roles of an Operating System (OS) is that of an illusionist. Briefly describe two illusions that an OS provides.)

Core Concepts

(topics will be covered in detail in the rest of the module)

- Processes
- Memory
- File systems and I/O
- Interfaces to the OS

Core Concept 1: Process

Process

- **An instance of a program running**

- A process has
  - Memory
  - Use of resources (I/O)

- Processes can exchange data
- Processes are created by other processes

Core Concept 1: Process

Why Processes?

1. **Multi-programming**

   Allows multiple programs to run at once

   And allows them to run '**securely**' in their own separate memory spaces – facilitates **protection**

   Processes can not interfere with other processes or with the OS

2. **Multi-user**

   Allow multiple users to use the OS at once

Core Concept 1: Process

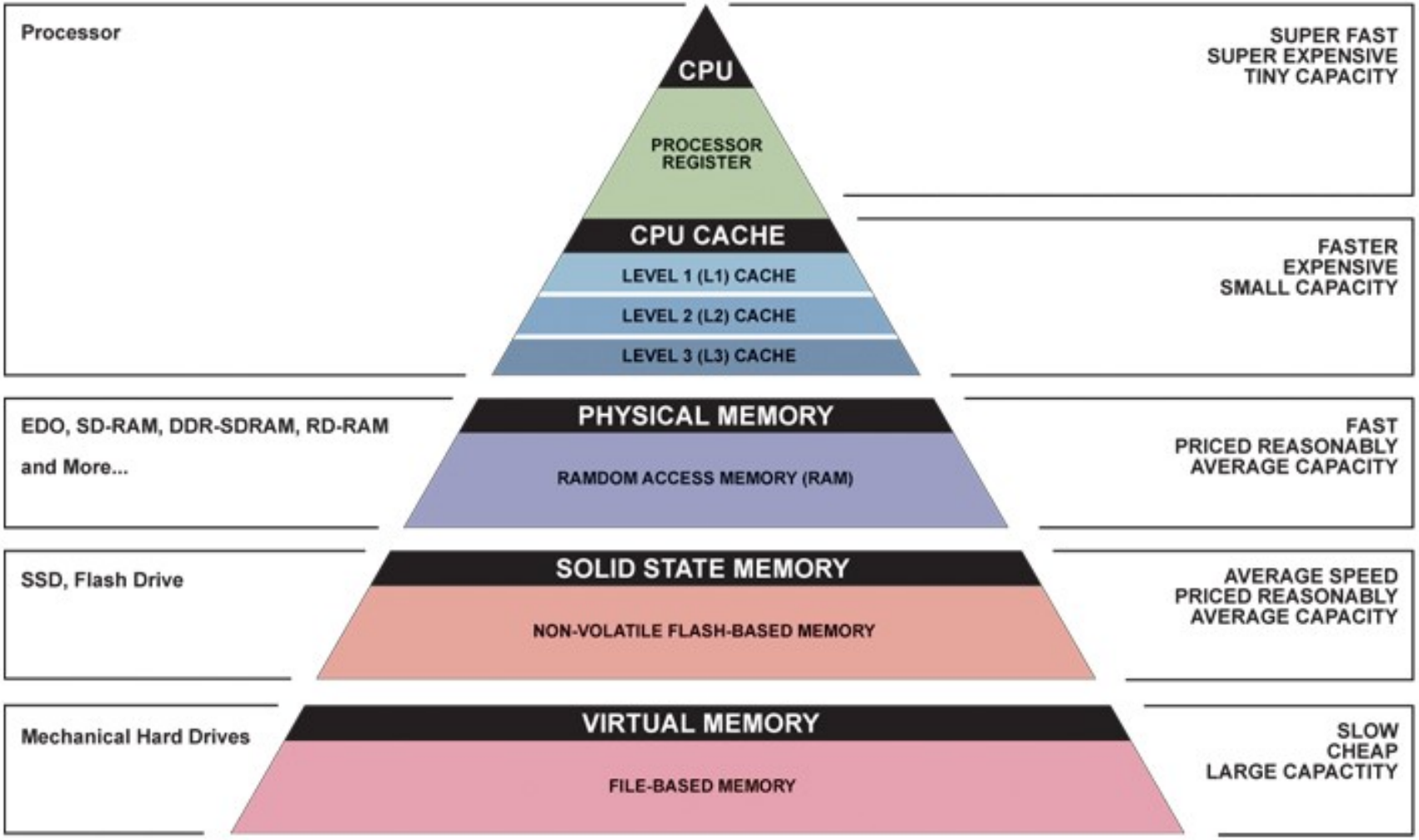
How Many Programs Run at Once?

- Uniprocessor – only one **really**
  - **Illusion of multiple processes**
  - Multiple processes increase throughput
  - Multiple processes allow timesharing (switching)
- Multiprocessors – several programs at once
  - Even our phones are **multi-core** but in a way it becomes irrelevant, the list of processes typically running is longer than the number of processors
- Concurrency vs. parallelism
  - **Parallel programming**: distribute a single task across multiple processors in order to go faster
  - **Concurrency**: play music while editing your code

Core Concept 2: Memory

Memory Management

- Data in cache, main memory or on disk?



Core Concept 2: Memory

Memory – Who?

- Lots of processes ➔ **sharing memory**
- Who gets the memory?
- Where in memory is my program?
- How much memory does my program get?

- **Virtual memory**
  - Memory as it appears to the program
  - **Illusion: your program can use all the memory**

- **Physical memory**
  - What is really happening underneath

Core Concept 3: Files & I/O

File Systems

- Files organised into directories
  - Hierarchy
- Ownership and protection (chmod 644 hello.txt)
- Attributes stored for files
  - Name, Size, Dates, etc.
  - There can be differences across OSs in what attributes they store for files or in what format
- **Illusion: disk is organised into files**
  - In reality disk only contains 0/1s and the OS has to logically organise things in such a way that one part of the disk stores file A, another part file B, etc.

Core Concept 3: Files & I/O

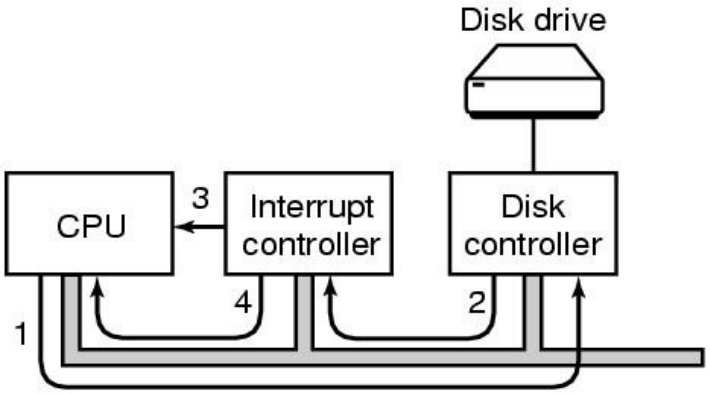
I/O e.g. Disk

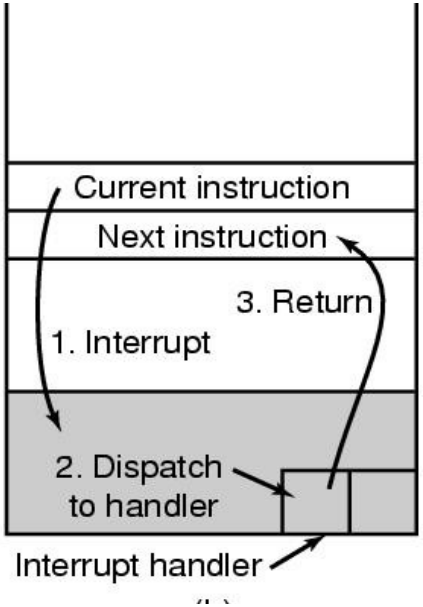
- Disk spins
  - Latency
- Assume a disk rotating at **6,000rpm** and a CPU executing **10^9 instructions per second**
- **Find:** How many instructions will be executed in the time it takes the disk to rotate once
- Each rotation ~10ms
- **Is 10ms a long time?**
  - ~10^7 instructions in the time it takes the disk to rotate once

- Operating system functions
  - **Throughput**: keeps busy while waiting (what shall I do while waiting for the disk?)- **Scheduling**
  - **File formats:** how information is arranged on disk

I/O – Interrupts

- Interrupt
  - Avoid waiting for response – **it is a signal to the processor** indicating an event that needs immediate attention
  - Process requests
  - OS interfaces to device – DRIVER
  - Device interrupts OS





Interfaces To The Operating System

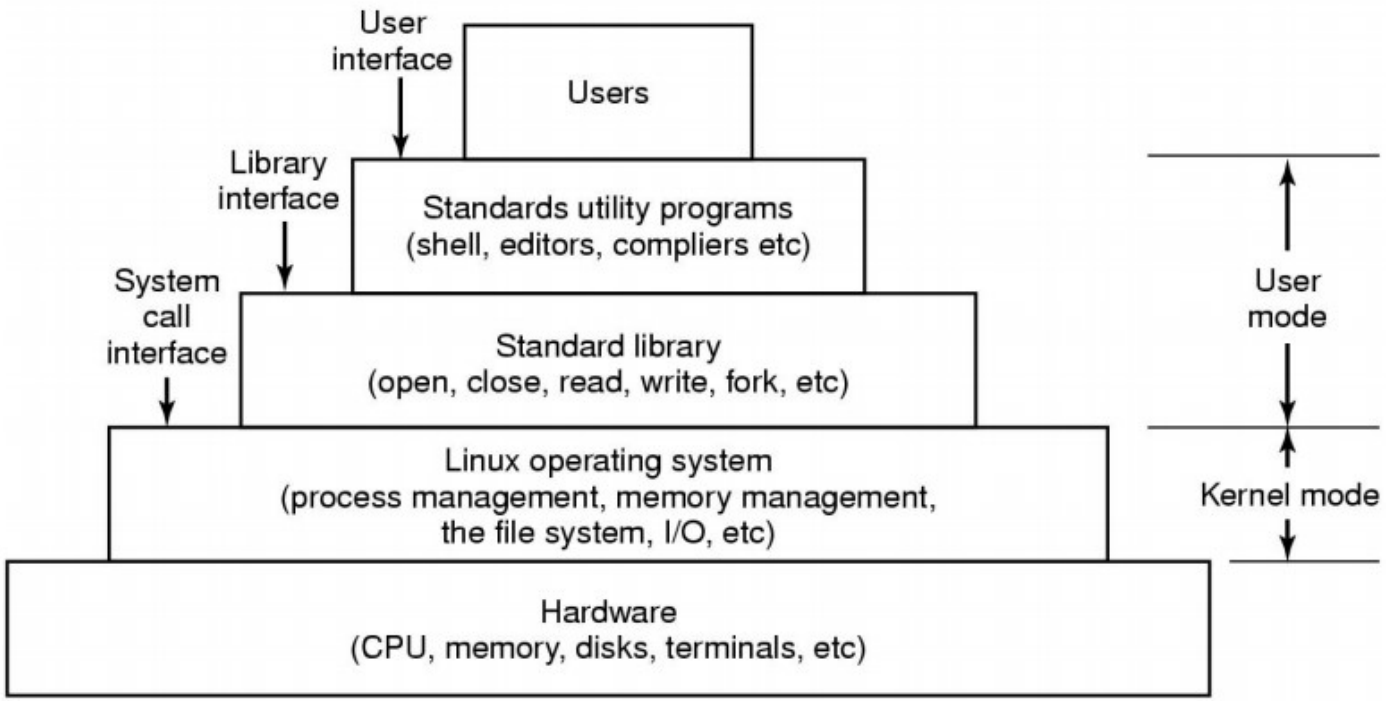


Figure taken from Tanenbaum

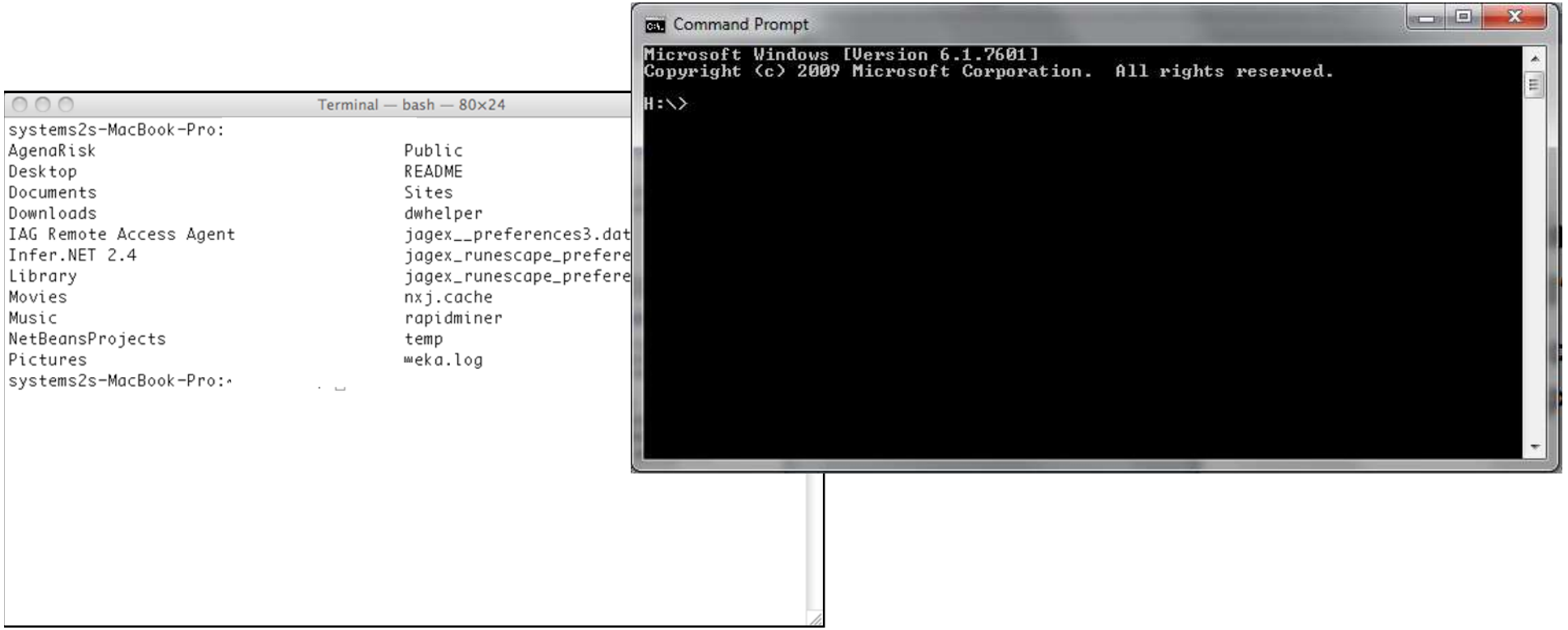
GUI

Shell(s)

System calls

Command Line

- Command line interface to Operating System
  - Unix 'shell'
  - Windows command prompt
  - Windows Powershell



Unix Shell

- Simple commands
  - Files: ls, rm, mkdir, rmdir, …
- Pipes and command redirection
- Programming language-like capability
  - Many shells: sh, csh
  - 'bash': (Bourne-again shell, pun intended) http://en.wikipedia.org/wiki/Bash_%28Unix_shell%29
  - **Scripting languages:** perl, php, python

Unix Command for Files

- ls *list the directory*
- mkdir *make a new directory*
- cd *change current directory*
- pwd *show current directory*
- rm *delete a file*
- rmdir *delete a directory*
- link *create a symbolic link*
- cp *copy a file*
- mv *move / rename a file*
- cat *look at a file*
- head *look at the start of a file*

Getting Information & Help

- ‘type’
- ‘help’
- ‘man’ pages
  - Describe each Unix command (and more)
  - Available online as well
- ‘info’ browser
  - Alternative to ‘man’
- Online, e.g.
  - http://linuxcommand.org/index.php

Scripting on this Module

- Php
  - You (mostly) know it (a bit)
  - Good support for POSIX
  - Standalone interpreter
- Main purpose
  - POSIX primitives
- Useful skill but I will minimise ‘php programming’ required
  - Templates provided, you extend the code

Php example

- Hello world, as a script

```
<?php
/*
 * Must be inside the <?php tag
 */
echo 'Hello World' ;
?>
```

System Calls

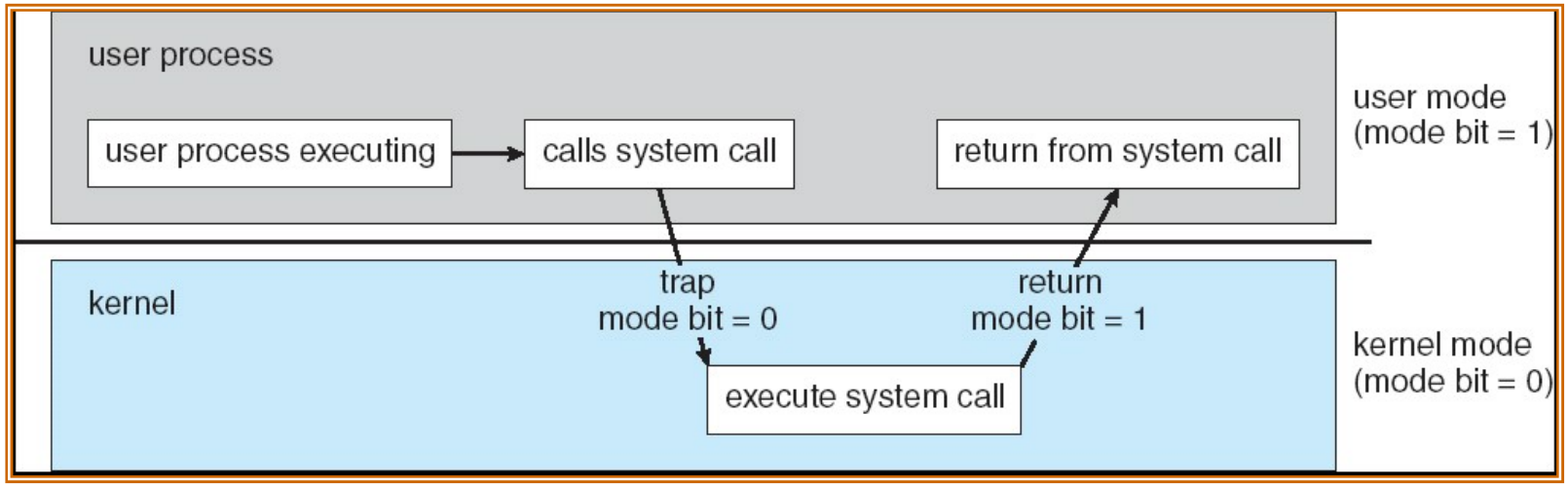
- **Programming interface to OS**
  - **'Call' the OS from your program, a library of functions** that you can use from your own programs
- POSIX
  - IEEE standard (minimum set of (mostly) system calls that any variant of UNIX must support)
- Example
  - 'read' bytes from a file
  - fd – file descriptor
  - buffer – place for results

```
Count = read(fd, buffer, nbytes)
```

http://linux.die.net/man/2/read

Systems Calls: What the OS Does (for a Programmer)

- Types of system calls
  - Process management, file management, device management, information management, communication
- Systems calls used in programs through functions
  - You do not use the call directly, you use a function that evokes a system call
  - Many 'shell commands' correspond to system calls
- **A system call is one way to switch from user mode to kernel mode**



Example System Calls

File management

Call	Description
fd = open(file, how, ...)	Open a file for reading, writing, or both
s = close(fd)	Close an open file
n = read(fd, buffer, nbytes)	Read data from a file into a buffer
n = write(fd, buffer, nbytes)	Write data from a buffer into a file
position = lseek(fd, offset, whence)	Move the file pointer
s = stat(name, &buf)	Get a file's status information

Process management

Call	Description
pid = fork( )	Create a child process identical to the parent
pid = waitpid(pid, &statloc, options)	Wait for a child to terminate
s = execve(name, argv, environp)	Replace a process' core image
exit(status)	Terminate process execution and return status

Other 'categories' of system calls:

Device management (read/write, get/set device attributes, …),
Information management (get/set time, get/set file attributes…),
Communication (send/receive messages, create/delete communication connection, …)

UNIX / Windows Win32 API

UNIX	Win32	Description
fork	CreateProcess	Create a new process
waitpid	WaitForSingleObject	Can wait for a process to exit
execve	(none)	CreateProcess = fork + execve
exit	ExitProcess	Terminate execution
open	CreateFile	Create a file or open an existing file
close	CloseHandle	Close a file
read	ReadFile	Read data from a file
write	WriteFile	Write data to a file
lseek	SetFilePointer	Move the file pointer
stat	GetFileAttributesEx	Get various file attributes
mkdir	CreateDirectory	Create a new directory
rmdir	RemoveDirectory	Remove an empty directory
link	(none)	Win32 does not support links
unlink	DeleteFile	Destroy an existing file
mount	(none)	Win32 does not support mount
umount	(none)	Win32 does not support mount
chdir	SetCurrentDirectory	Change the current working directory
chmod	(none)	Win32 does not support security (although NT does)
kill	(none)	Win32 does not support signals
time	GetLocalTime	Get the current time

Summary - What Is An Operating System?

- Kernel
  - Control of processes
  - Hardware I/O drivers
  - Memory management
  - Kernel mode vs. user mode
- Services and utilities
  - File systems
  - Network interface
- System call: programmers interface
- Command shell, GUI shell
- Possibly more (or less) – no universally agreed 'definition'

Summary - OS is about Illusions

- Several programs are running simultaneously
- My program has all the memory
- Disk is organised in files
- Disk access is quick
- Storage devices work the same
- … etc.

OS creates an 'ideal' computer from a real one