CPE 460 Operating System Design

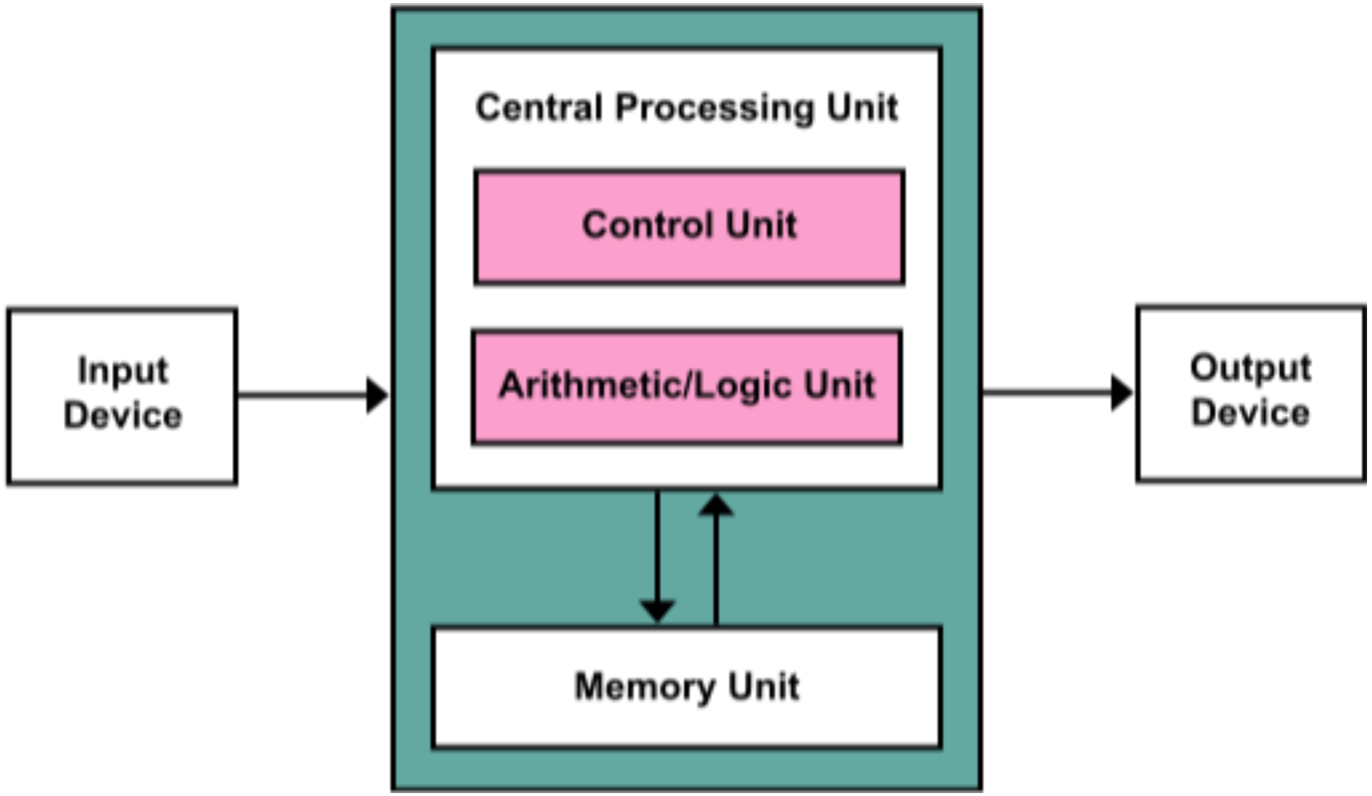
Lecture 2: Operating Systems Structures

Ahmed Tamrawi

February 13, 2017

1945 Von Neumann Architecture

John von Neumann



Computer System Architecture

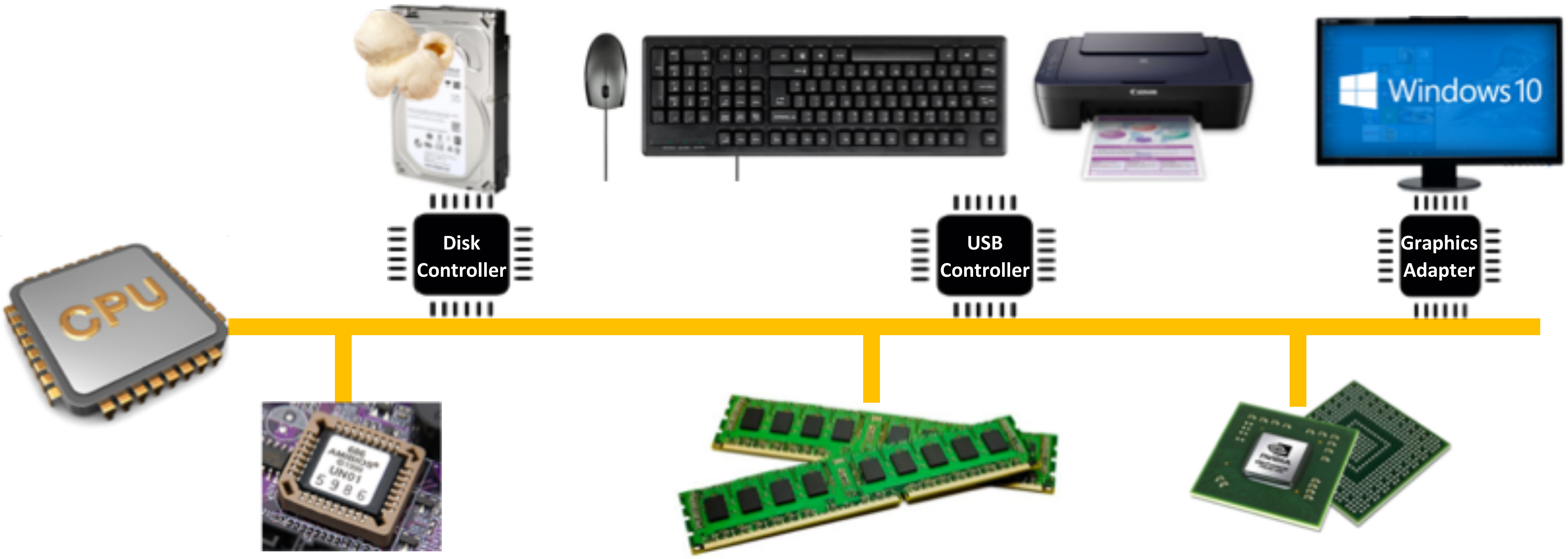
Single Processor System

Clustered System

Multiprocessor System

Single Processor System

One main CPU capable of executing a **general-purpose** instruction set

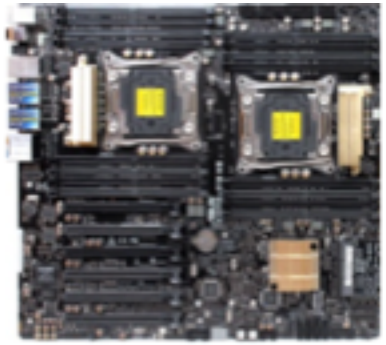


Multiprocessor System

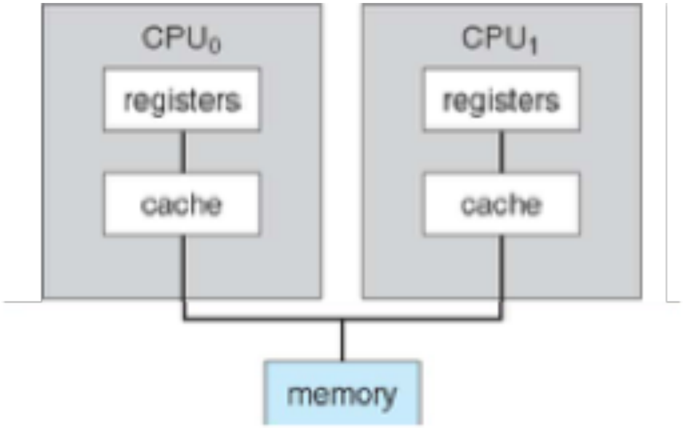
Asymmetric Multiprocessing

Two or more processors in close communication, sharing the computer bus and sometimes the clock, memory, and peripheral devices.

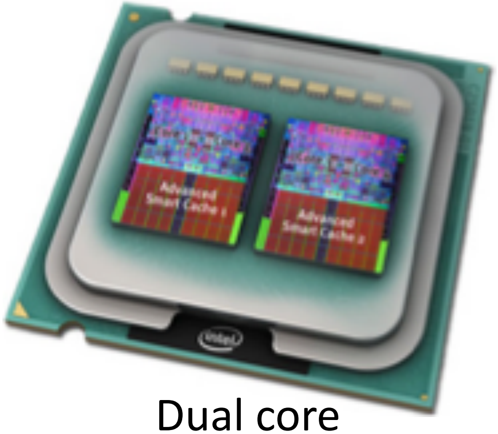
Symmetric Multiprocessing



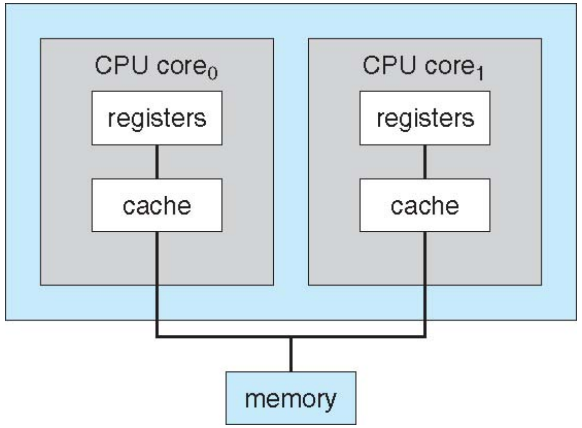
Dual Processor



- Parallelization
- Increased Throughput
- Economy of Scale
- Increased Reliability
- Graceful Degradation
- Fault Tolerant



Dual core



Multiprocessor System

Two or more processors in close communication, sharing the computer bus and sometimes the clock, memory, and peripheral devices.

Asymmetric Multiprocessing

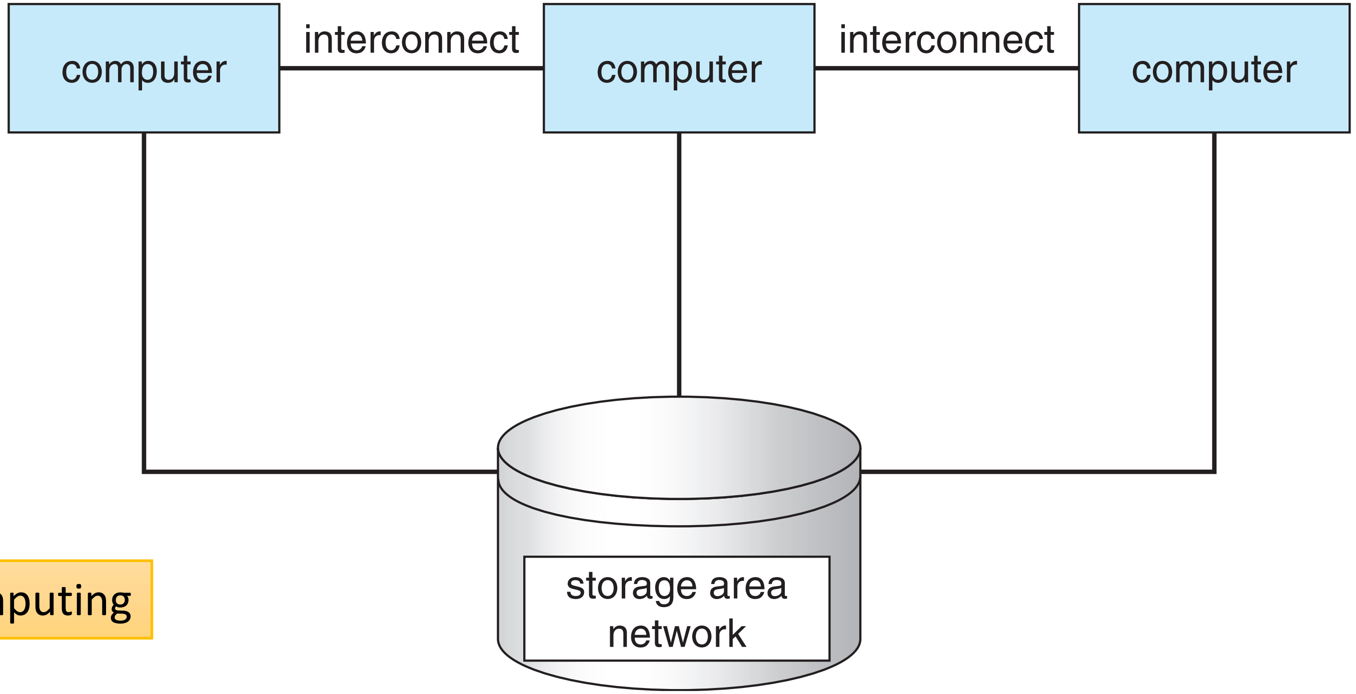
Symmetric Multiprocessing

Clustered System

Multiple systems working together

Asymmetric Clustering

Symmetric Clustering



High-Performance Computing

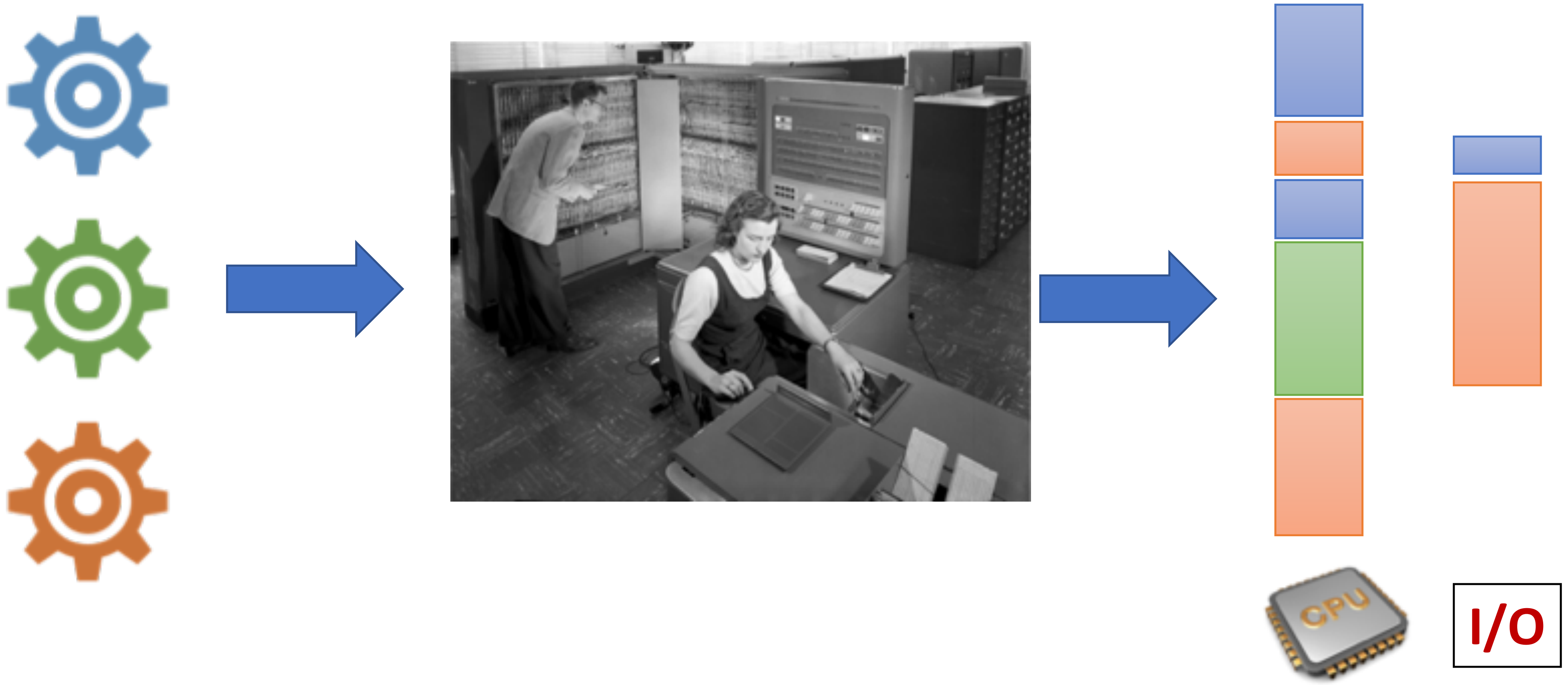
High Availability

Computer System Structure

Multiprogramming (Batch System)

Timesharing (Multitasking)

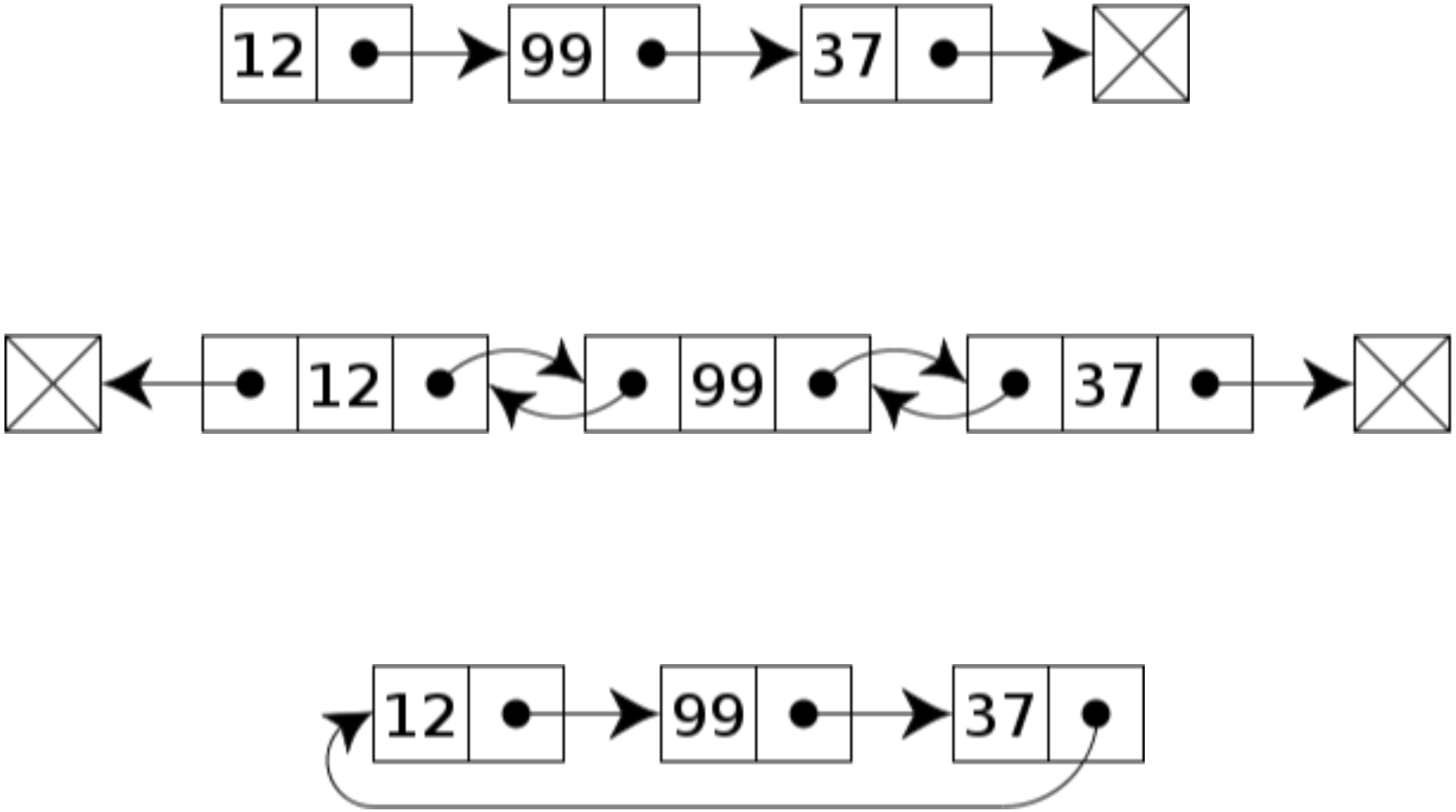
Multiprogramming (Batch System)



Timesharing (Multitasking)

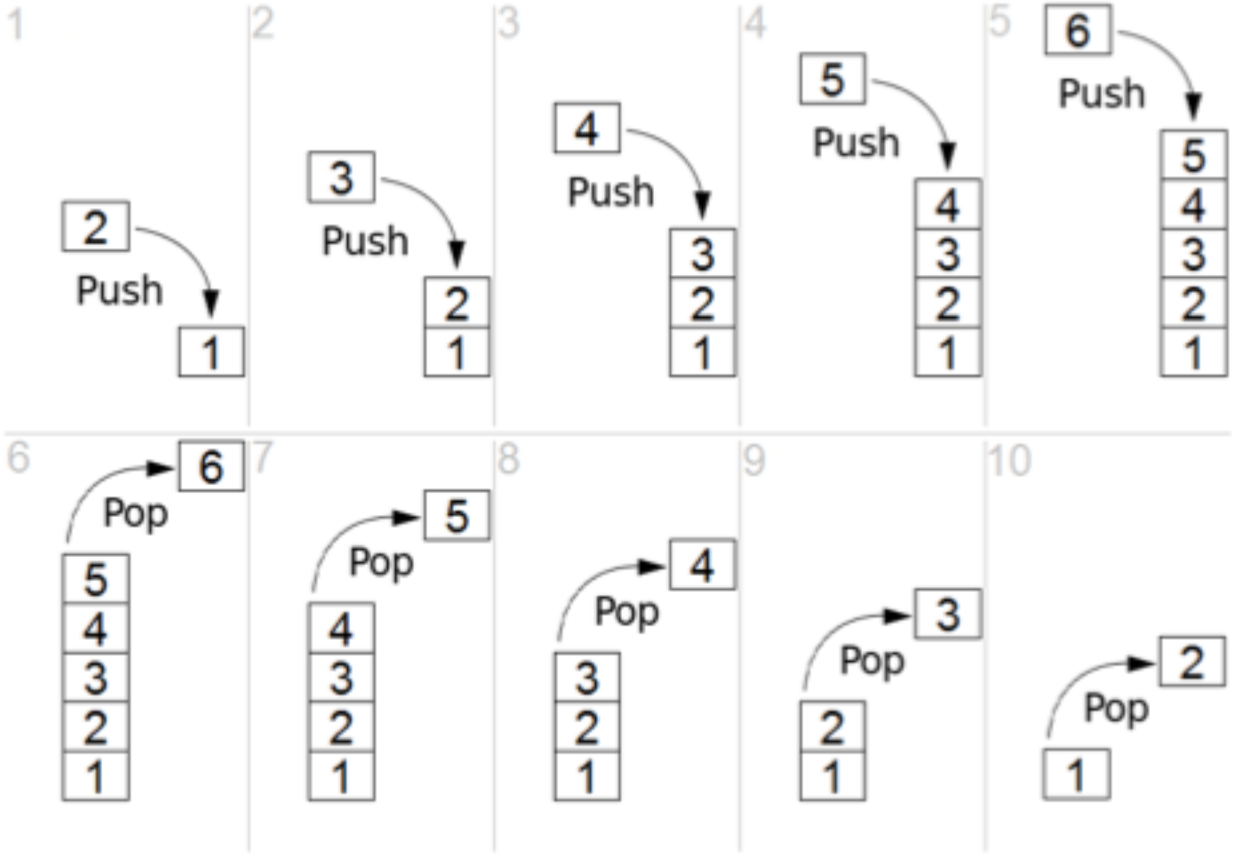
Kernel Data Structures

Linked List



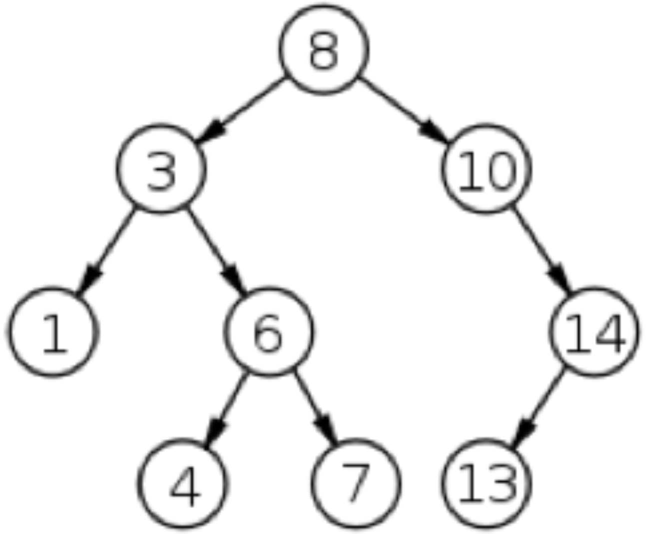
https://en.wikipedia.org/wiki/Linked_list

Stack



https://en.wikipedia.org/wiki/Stack_(abstract_data_type)

Binary Search Trees



Binary search tree

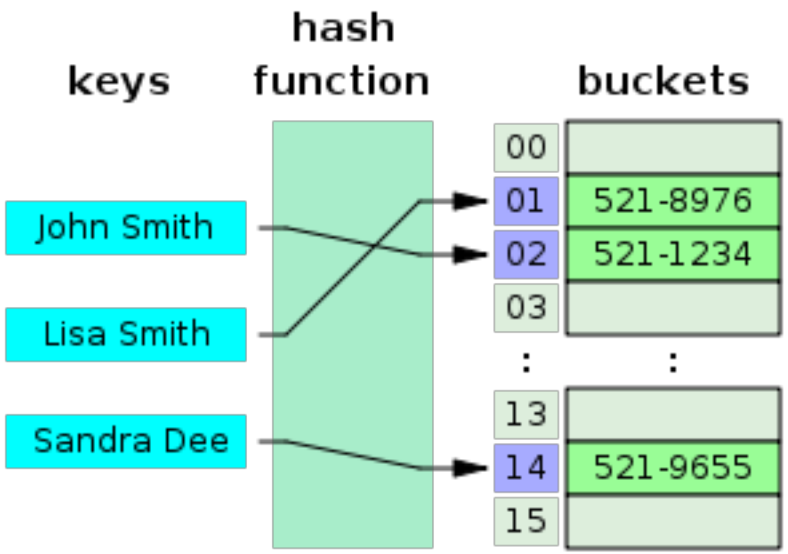
Type	tree
Invented	1960
Invented by	P.F. Windley, A.D. Booth, A.J.T. Colin, and T.N. Hibbard

Time complexity in big O notation

Algorithm	Average	Worst Case
Space	$\Theta(n)$	$O(n)$
Search	$\Theta(\log n)$	$O(n)$
Insert	$\Theta(\log n)$	$O(n)$
Delete	$\Theta(\log n)$	$O(n)$

https://en.wikipedia.org/wiki/Binary_search_tree

Hash Table



Hash table

Type	Unordered associative array	
Invented	1953	
Time complexity in big O notation		
Algorithm	**Average**	**Worst Case**
Space	$O(n)$[1]	$O(n)$
Search	$O(1)$	$O(n)$
Insert	$O(1)$	$O(n)$
Delete	$O(1)$	$O(n)$

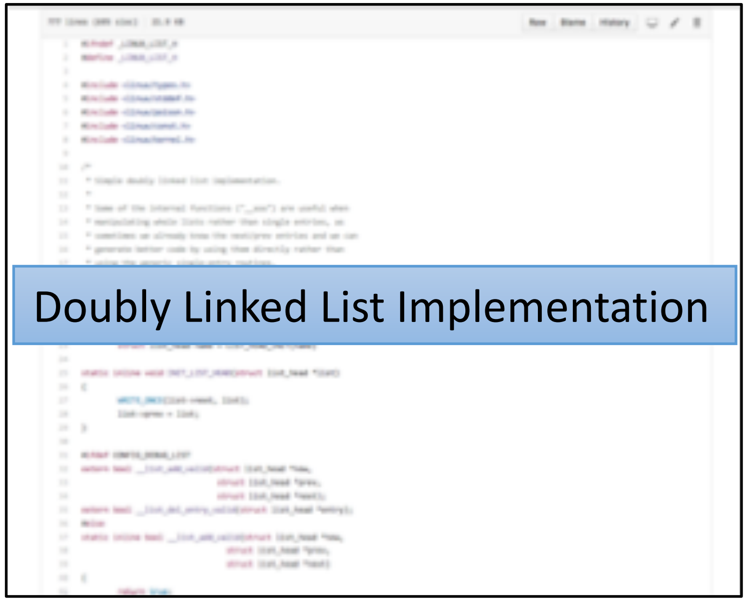
Bitmap

A string of *n* binary digits representing the status of *n* items

10101111000111 0101

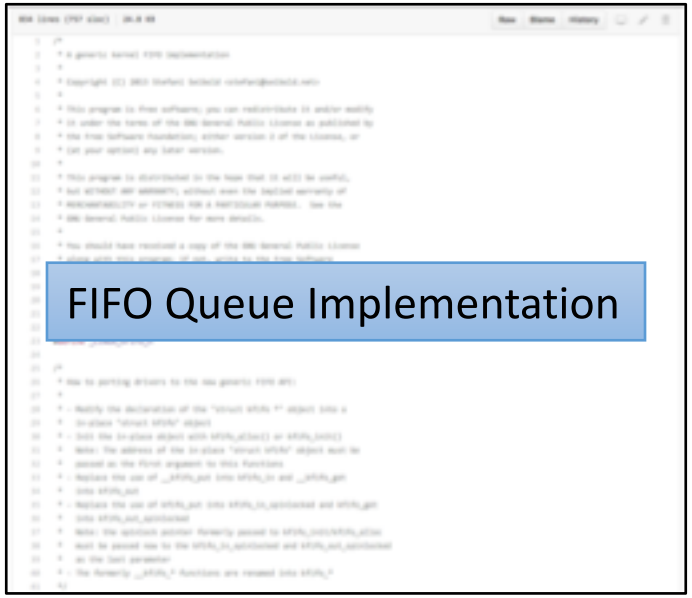
Linux Travolds

Linux Kernel Repository
https://github.com/torvalds/linux



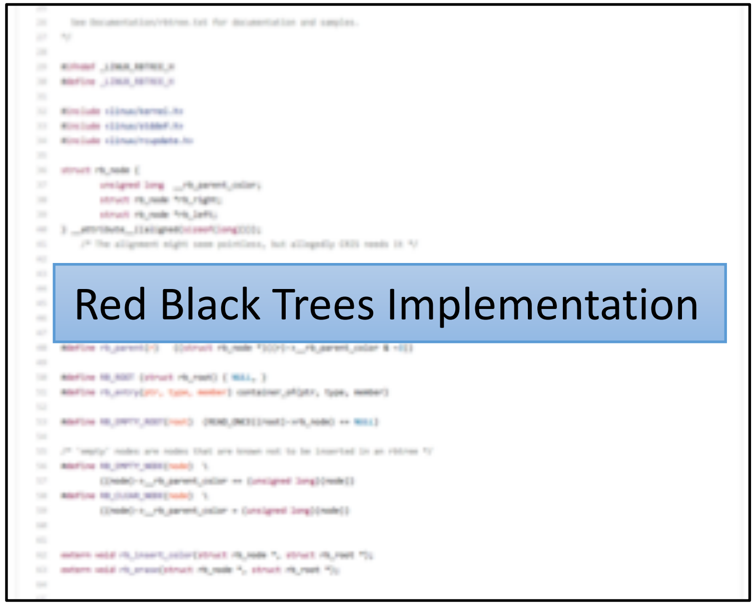
Doubly Linked List Implementation

https://github.com/torvalds/linux/blob/master/include/linux/list.h



FIFO Queue Implementation

https://github.com/torvalds/linux/blob/master/include/linux/kfifo.h

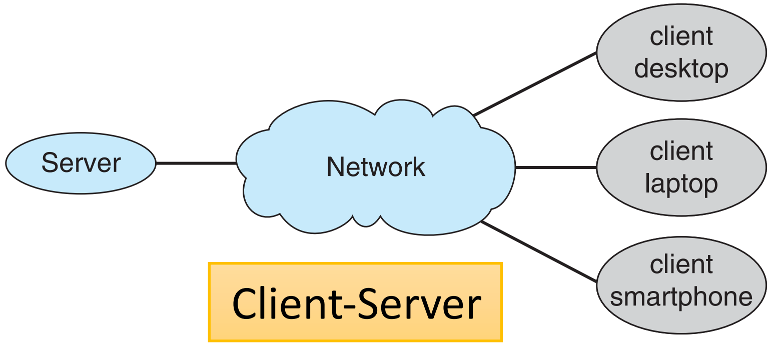


Red Black Trees Implementation

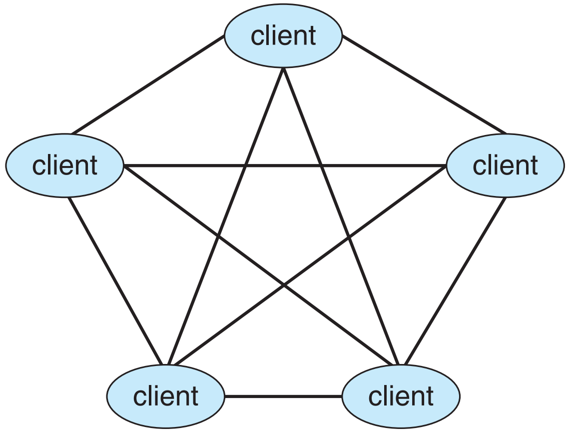
https://github.com/torvalds/linux/blob/master/include/linux/rbtree.h

Computing Environments

Traditional Computing



Client-Server



Peer-to-Peer

Mobile

Cloud Computing

Real-Time Embedded Systems

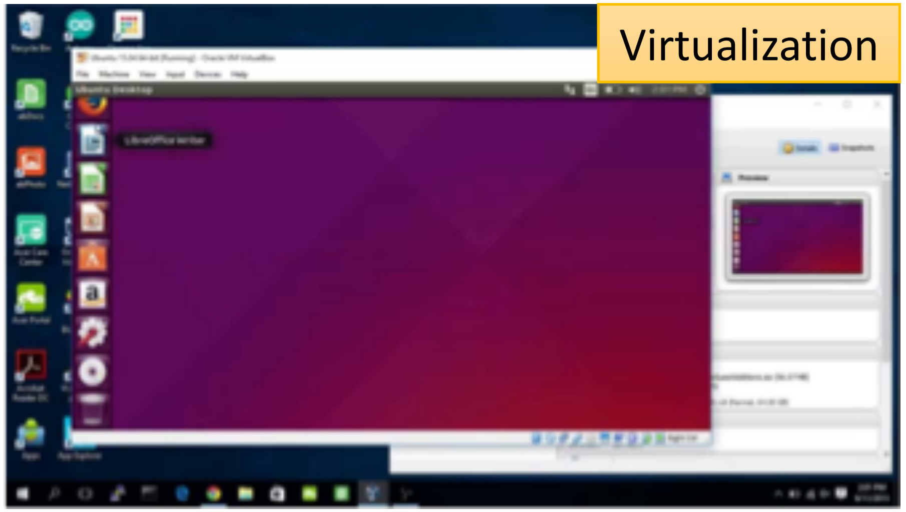
Emulation & Virtualization

Allows operating systems to run applications within other OSes

Emulation

Motorola 68k CPU

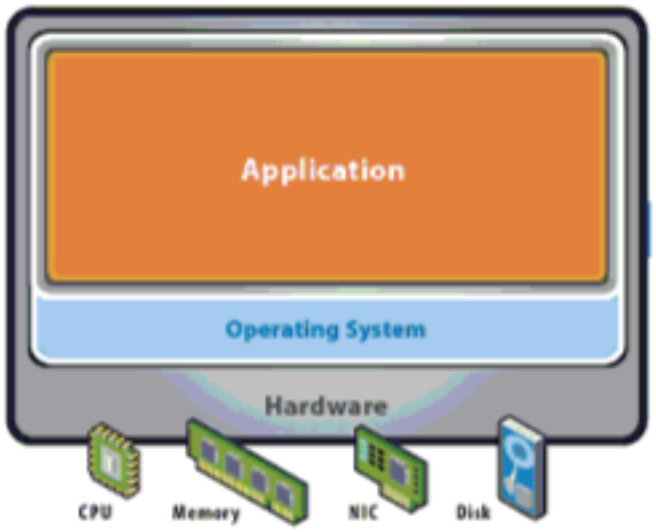
Intel x86 CPU



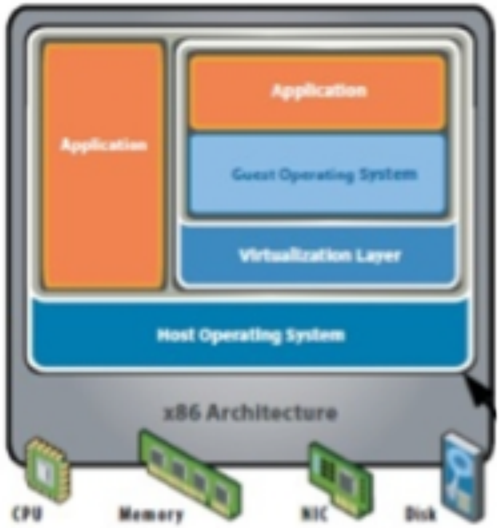
Virtualization

Emulation used when source CPU type different from target type

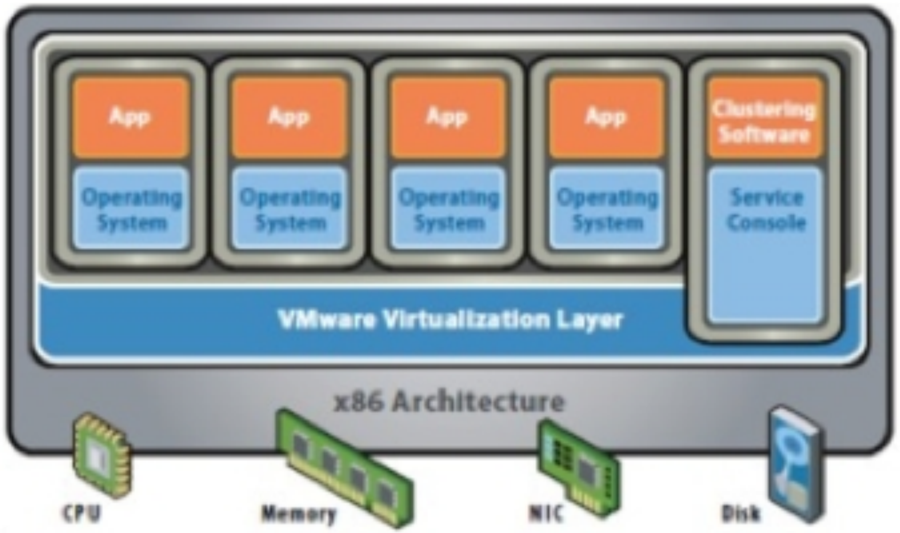
OS natively compiled for CPU, running guest OSes also natively compiled



No Virtualization



OS-Level Virtualization



Full Virtualization

open source

http://www.apple.com/opensource/

https://github.com/android

FreeBSD

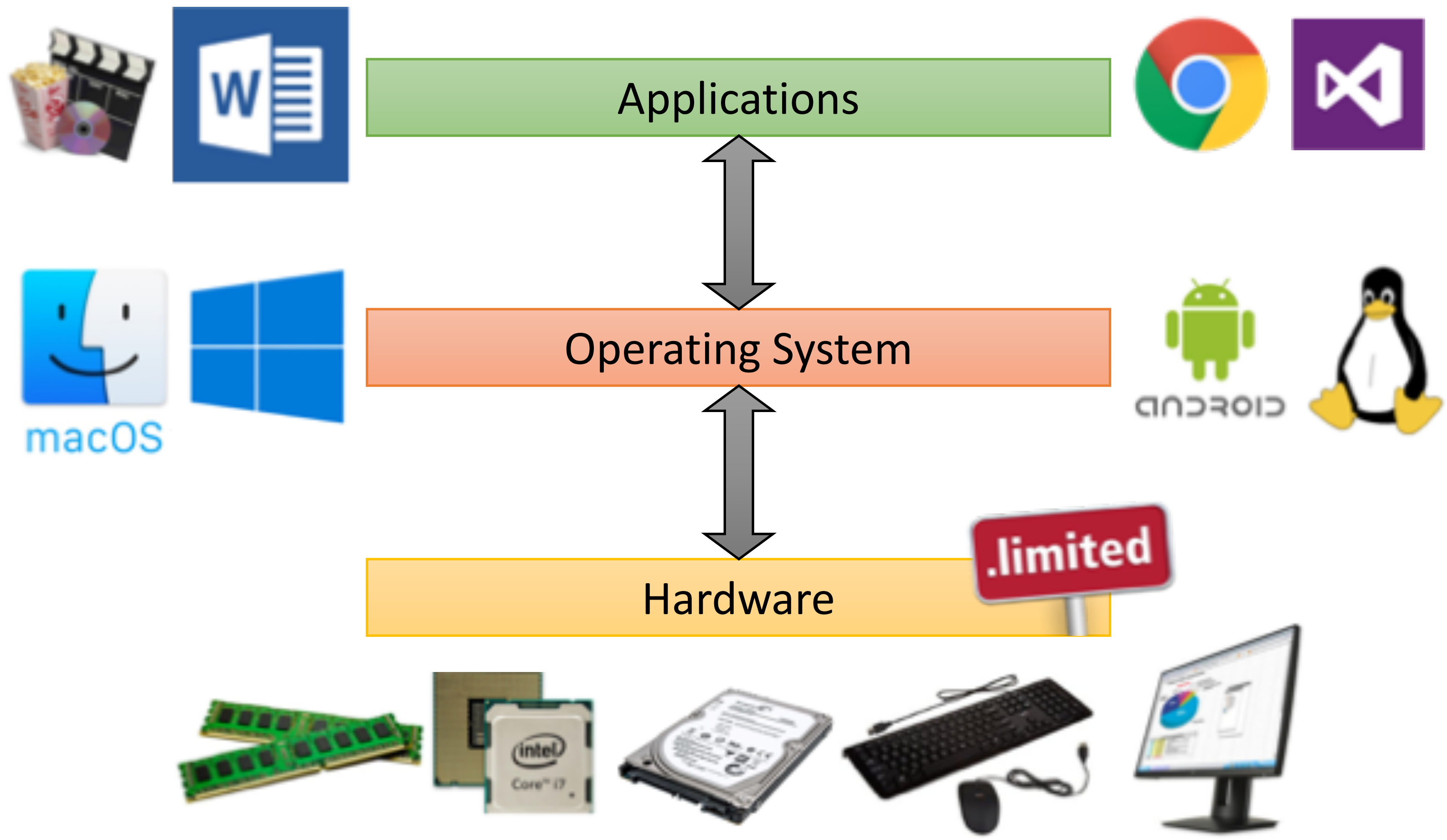
https://github.com/torvalds/linux

https://svnweb.freebsd.org/base/

Realistic View of Operating System

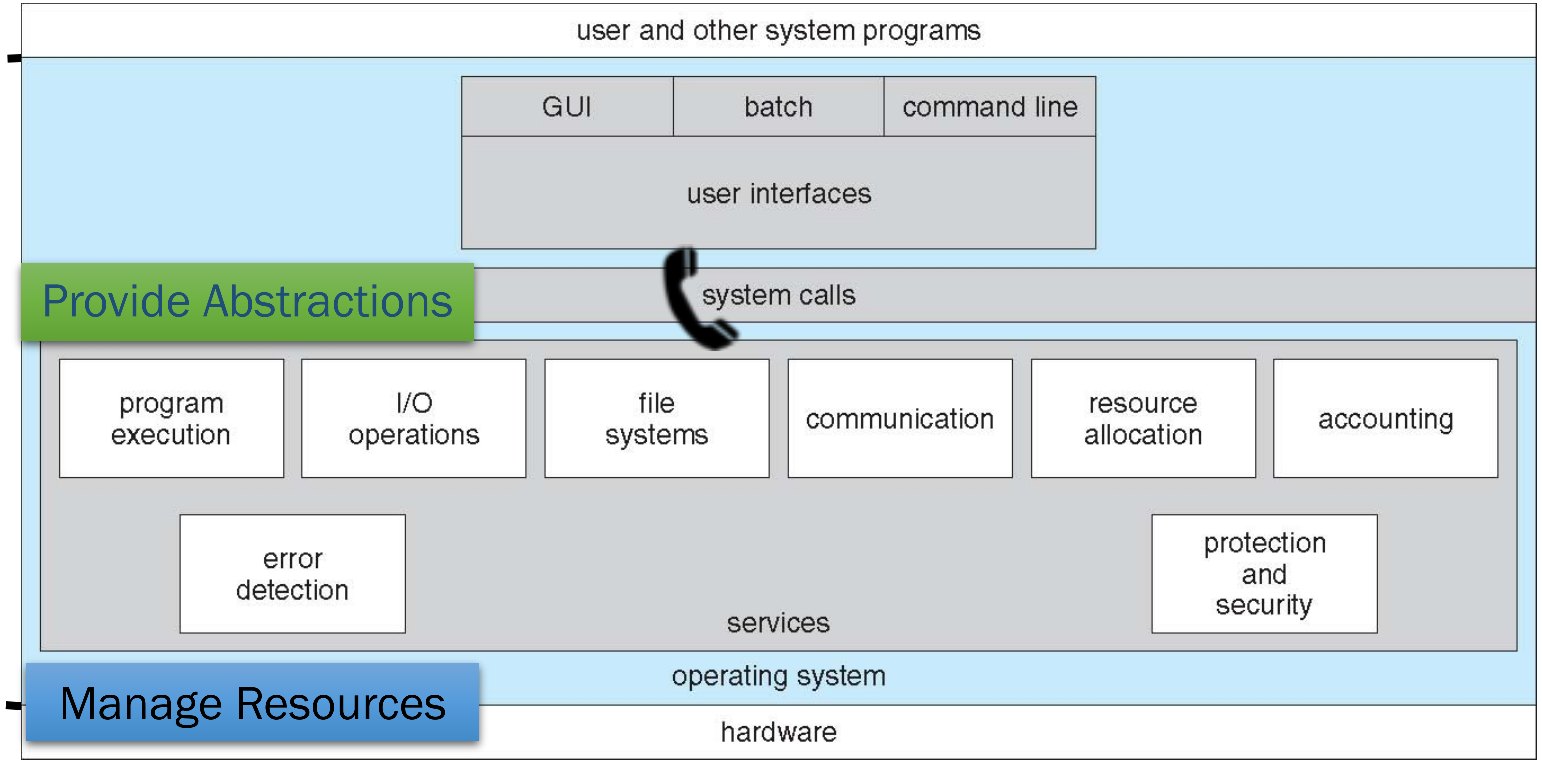
Manage Resources

Provide Abstractions



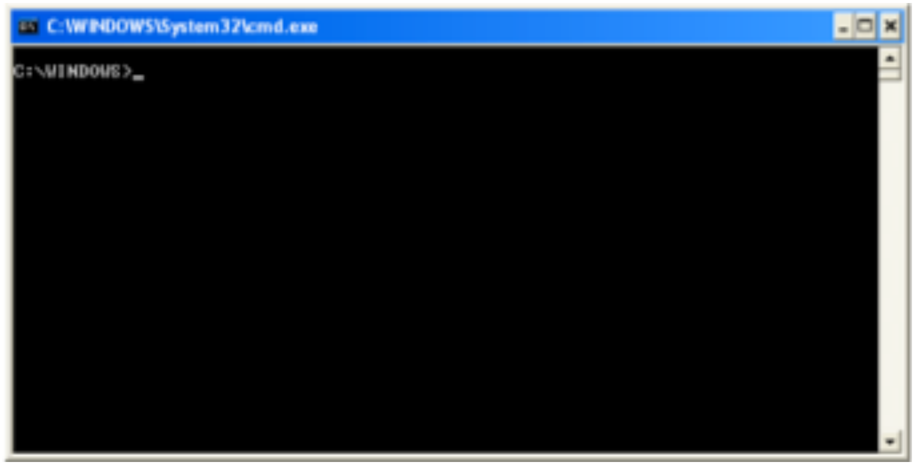
A View of Operating System Services

Operating systems provide an environment for execution of programs and services to programs and users



User Operating System Interface - CLI

CLI or command interpreter allows direct command entry



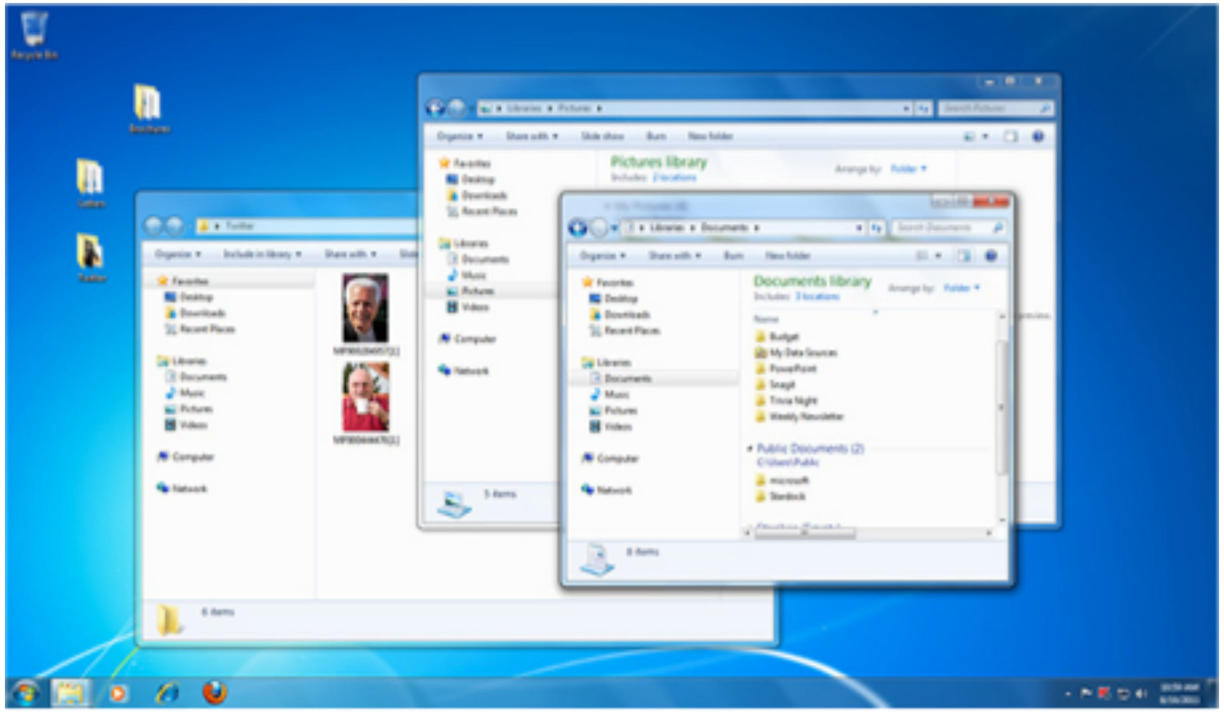
Sometimes implemented in kernel, sometimes by systems program

Primarily fetches a command from user and executes it

Sometimes commands built-in, sometimes just names of programs

User Operating System Interface - GUI

User-friendly desktop metaphor interface



Many systems now include both CLI and GUI interfaces

User Operating System Interface - Touchscreen Interfaces

Touchscreen devices require new interfaces

An operating system is interrupt driven

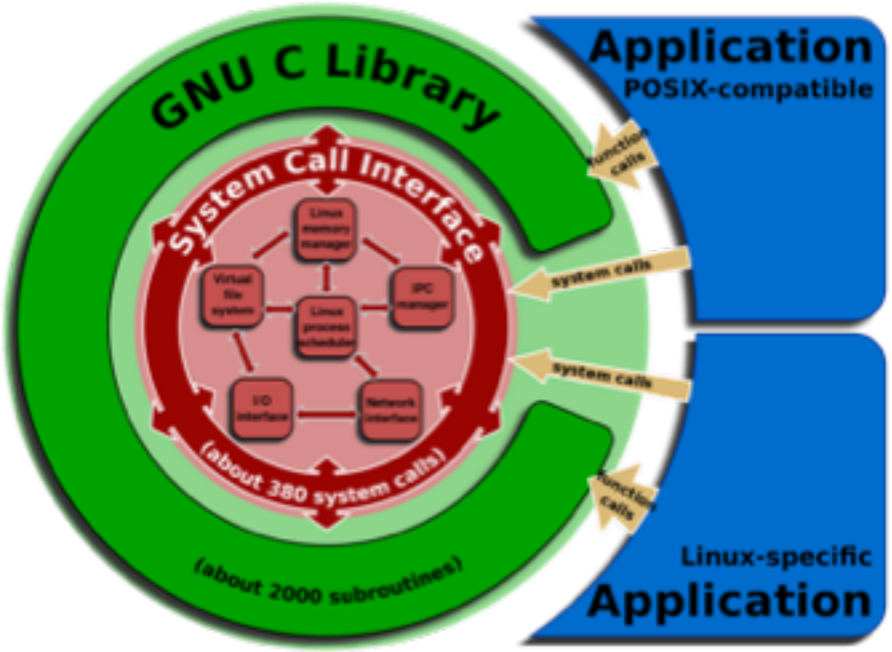
System Call

Software Interrupt (Trap)

Programming interface to the services provided by the OS

Typically written in a high-level language (C or C++)

Accessible via a high-level **Application Programming Interface (API)** rather than direct system call use



Create, Delete Communication Connection
Message Passing Model Host/Process Name
Shared-Memory Model
Transfer Status Information
Attach/Detach Remote Devices

Create/Terminate/Load/Execute Process
Get/Set Process Attributes
Wait for Time/Event
wait event, signal event
Allocate/Free/Dump Memory
Locks for Process Synchronization

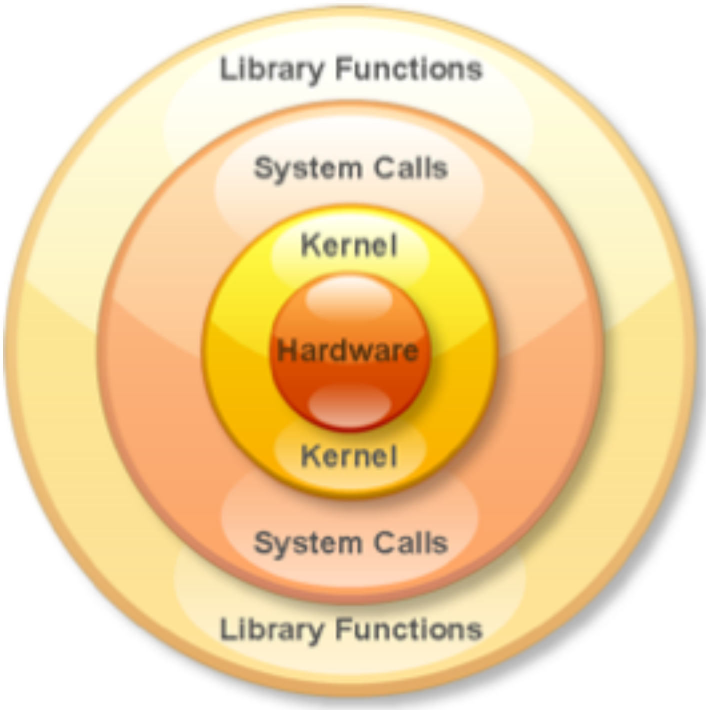
Control access to resources
Get and set permissions
Allow and deny user access

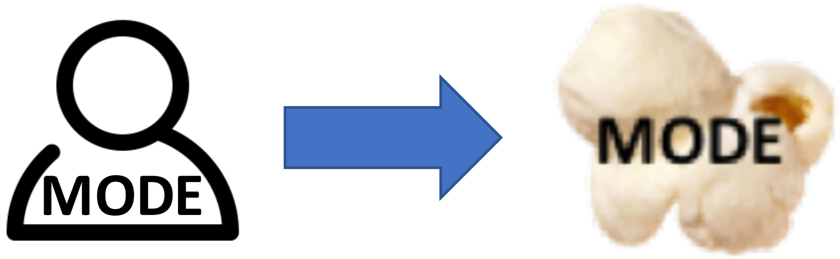
Create/Delete/Open/Close/Read/Write File
Get/Set File Attributes

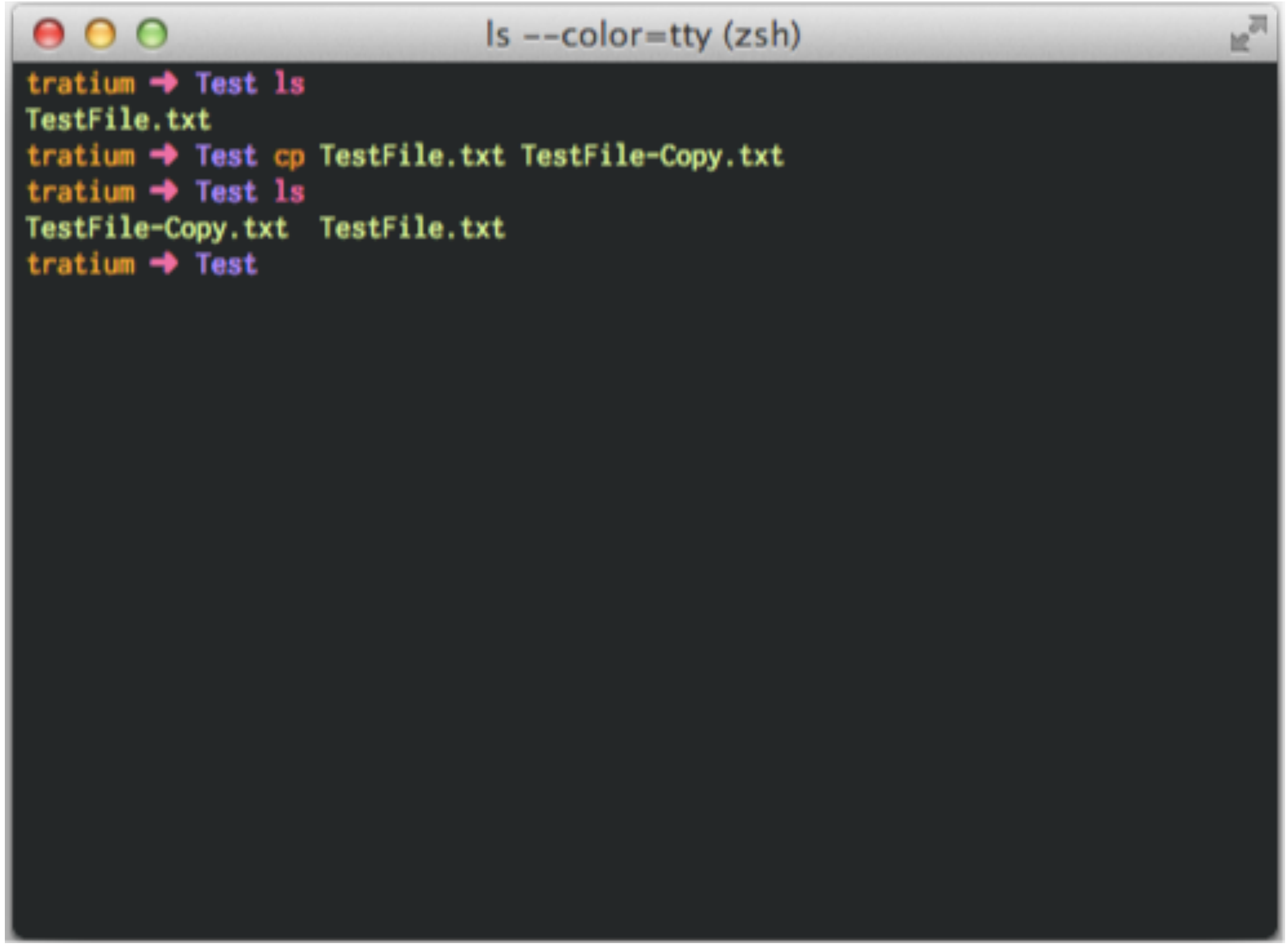
Get/Set Time or Date
Get/Set System Data

Request/Release/Read/Write Device
Get/Set Device Attributes
Logically Attach/Detach devices

User processes **cannot** perform *privileged operations* themselves



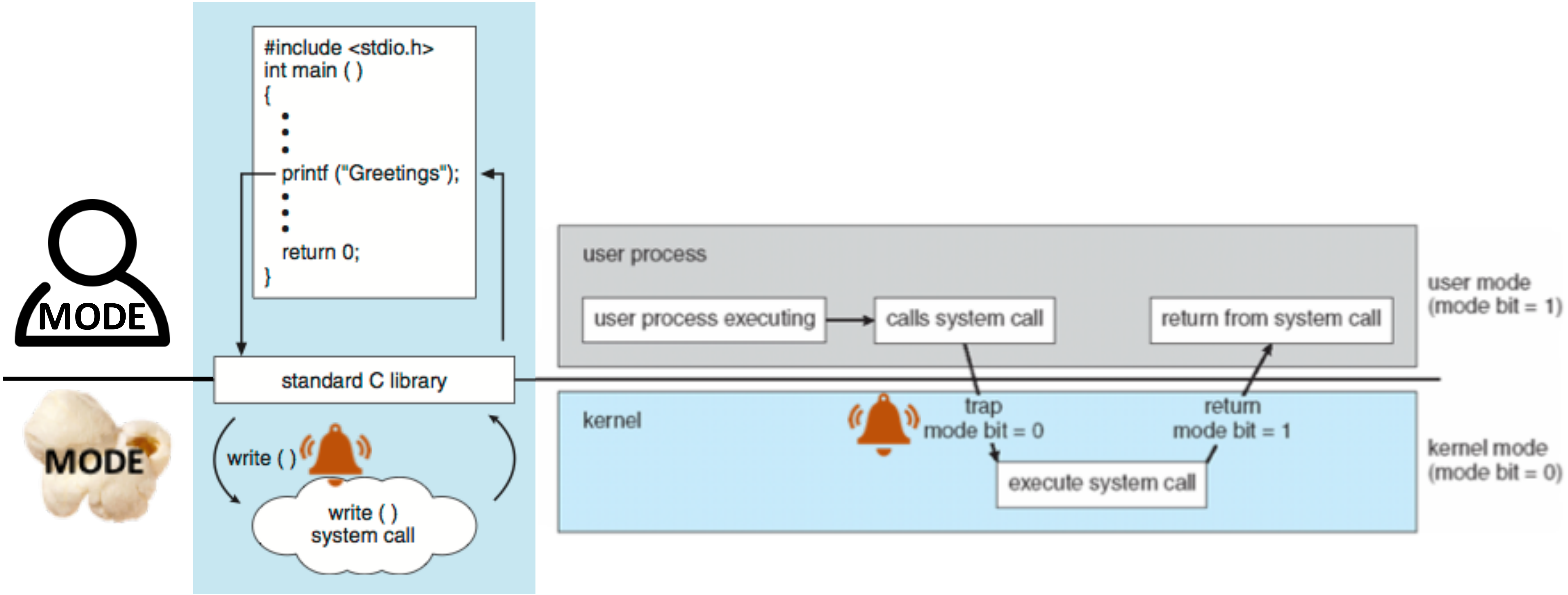





```
strace cp TestFile.txt TestFile-Copy.txt
```

1. Open the input file (TestFile.txt)
2. If (TestFile.txt) does not exist, abort
3. Create the output file (TestFile-Copy.txt)
4. If (TestFile-Copy.txt) exists, abort
5. Loop Until No bytes available in TestFile.txt
    1. Read byte from (TestFile.txt)
    2. Write byte to (TestFile-Copy.txt)
6. Close (TestFile.txt)
7. Close (TestFile-Copy.txt)
8. Terminate normally.

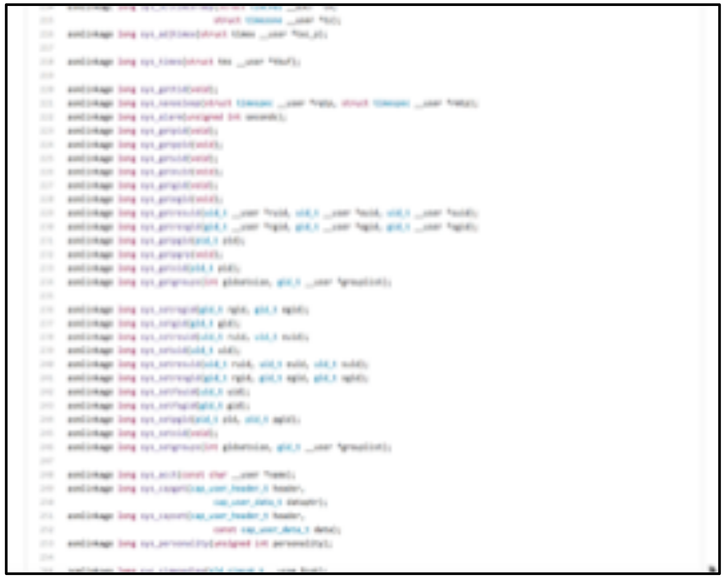
System call sequence to copy the contents of one file to another file


MODE
MODE
#include <stdio.h>
int main ( )
{
printf ("Greetings");
return 0;
}
standard C library
write ( )
write ( )
system call
user process
user process executing
calls system call
return from system call
user mode
(mode bit = 1)
kernel
trap
mode bit = 0
return
mode bit = 1
execute system call
kernel mode
(mode bit = 0)

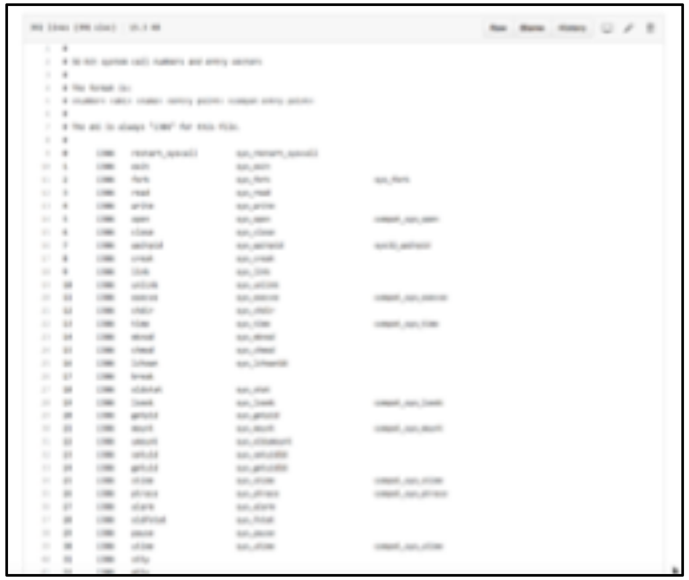
How the kernel know which system call to execute?

System call interface maintains a **System Call Table** (**Vector**) with a number corresponding to each system call

The system call interface invokes the intended system call in OS kernel and returns status of the system call and any return values

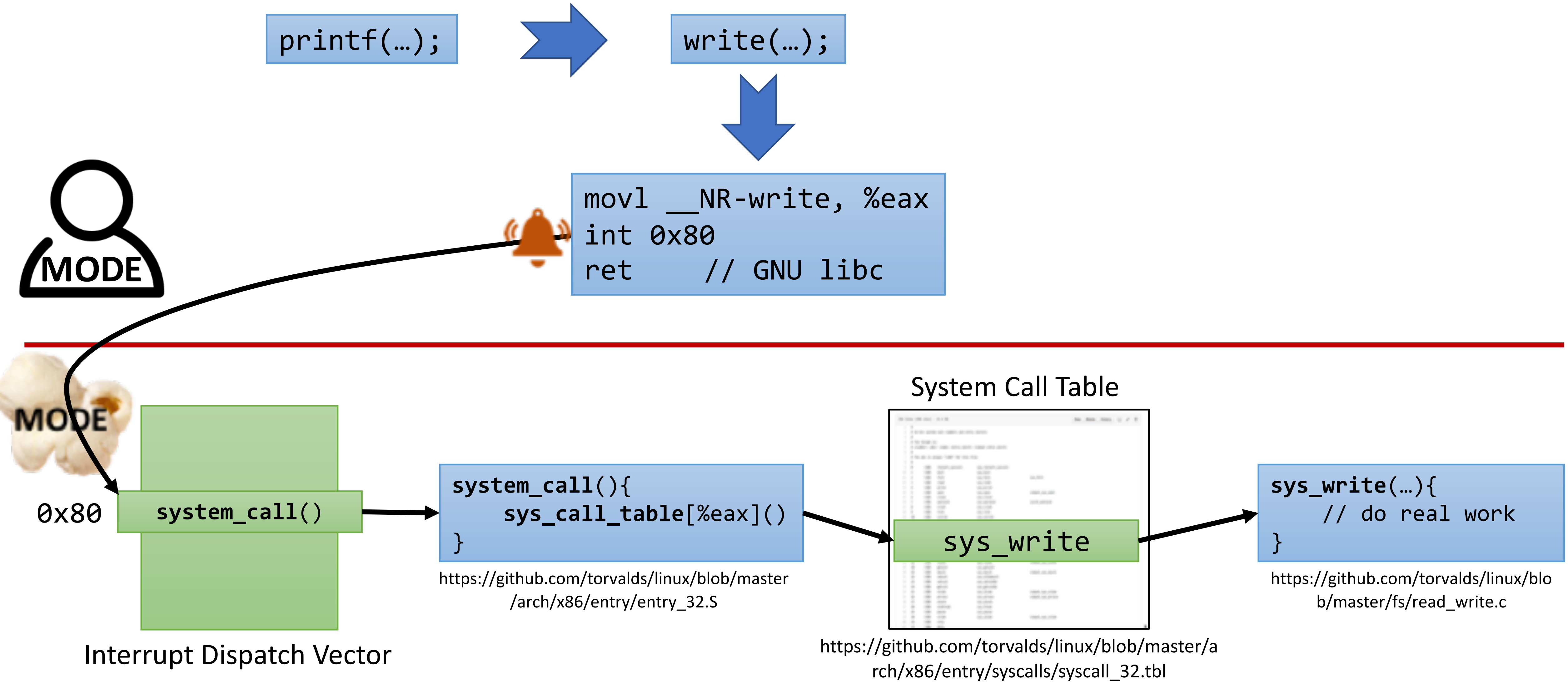


https://github.com/torvalds/linux/blob/master/include/linux/syscalls.h



https://github.com/torvalds/linux/blob/master/arch/x86/entry/syscalls/syscall_32.tbl

How the kernel know which system call to execute?



How Parameters are Passed?

Registers

More parameters than registers

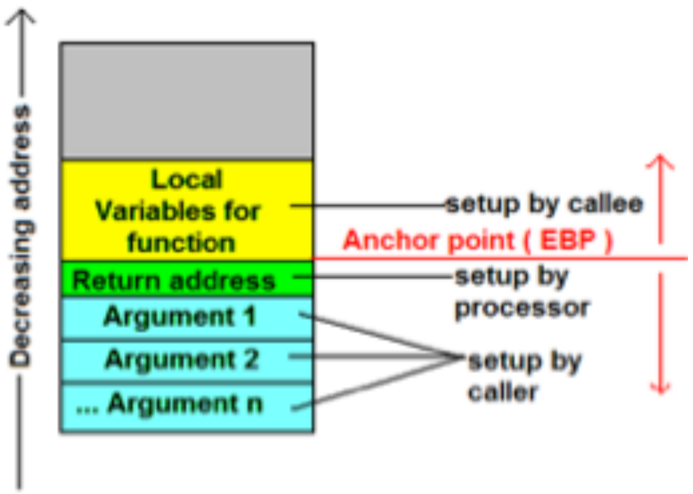
```
printf("Hello!");
```

```
write(screen, "Hello!" , …);
```

```
movl "Hello", %edx
movl __NR-write, %eax
int 0x80
ret      // GNU libc
```

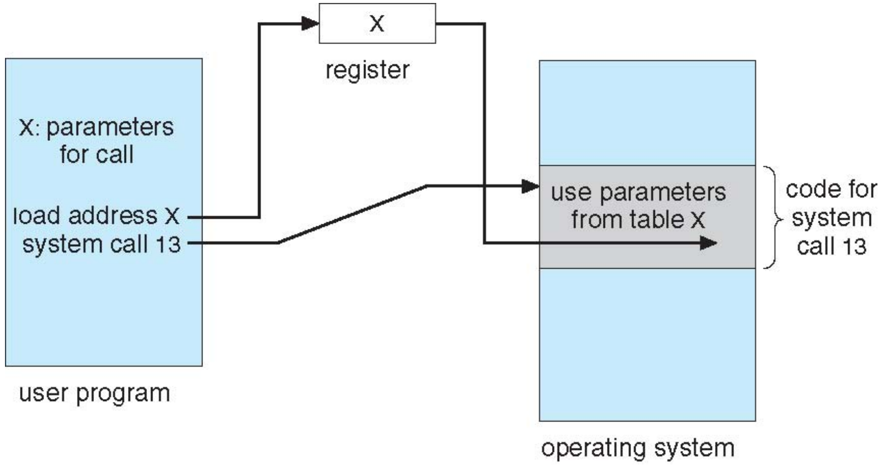
User-Mode Stack

Do not limit the number or length of parameters being passed



Block/Table in Memory

Do not limit the number or length of parameters being passed

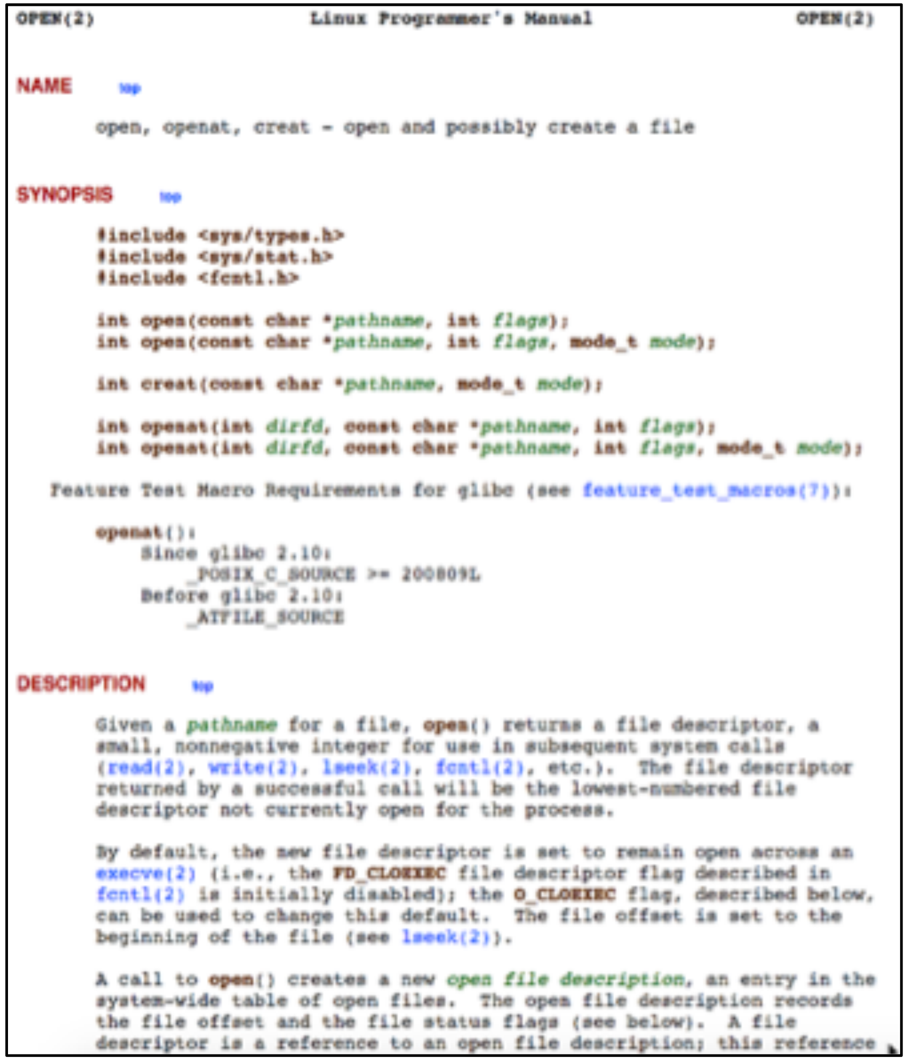


Open
System Call

The caller need know nothing about how the system call is implemented

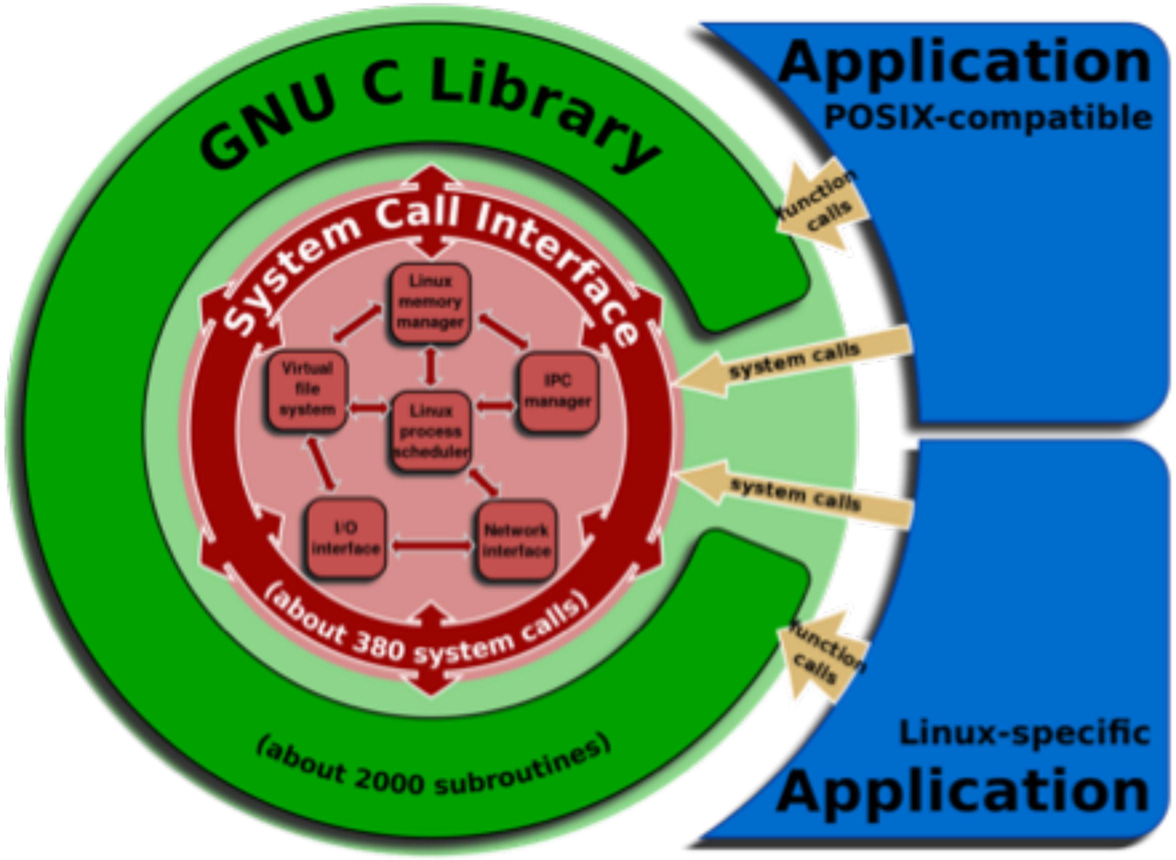
Just needs to obey API and understand what OS will do as a result call

Most details of OS interface hidden from programmer by API
Managed by run-time support library (set of functions built into libraries included with compiler)



```
OPEN(2)                 Linux Programmer's Manual                 OPEN(2)

NAME       top

       open, openat, creat - open and possibly create a file

SYNOPSIS       top

       #include <sys/types.h>
       #include <sys/stat.h>
       #include <fcntl.h>

       int open(const char *pathname, int flags);
       int open(const char *pathname, int flags, mode_t mode);

       int creat(const char *pathname, mode_t mode);

       int openat(int dirfd, const char *pathname, int flags);
       int openat(int dirfd, const char *pathname, int flags, mode_t mode);

   Feature Test Macro Requirements for glibc (see feature_test_macros(7)):

       openat():
           Since glibc 2.10:
               _POSIX_C_SOURCE >= 200809L
           Before glibc 2.10:
               _ATFILE_SOURCE

DESCRIPTION       top

       Given a pathname for a file, open() returns a file descriptor, a
       small, nonnegative integer for use in subsequent system calls
       (read(2), write(2), lseek(2), fcntl(2), etc.).  The file descriptor
       returned by a successful call will be the lowest-numbered file
       descriptor not currently open for the process.

       By default, the new file descriptor is set to remain open across an
       execve(2) (i.e., the FD_CLOEXEC file descriptor flag described in
       fcntl(2) is initially disabled); the O_CLOEXEC flag, described below,
       can be used to change this default.  The file offset is set to the
       beginning of the file (see lseek(2)).

       A call to open() creates a new open file description, an entry in the
       system-wide table of open files.  The open file description records
       the file offset and the file status flags (see below).  A file
       descriptor is a reference to an open file description; this reference
```

http://man7.org/linux/man-pages/man2/open.2.html




```
#include <unistd.h>
#include <fcntl.h>

int main() {
    int filedesc = open("testfile.txt", O_WRONLY | O_APPEND);
    if(filedesc < 0){
        return 1;
    }
    return 0;
}
```

With GNU C Library (Glibc)

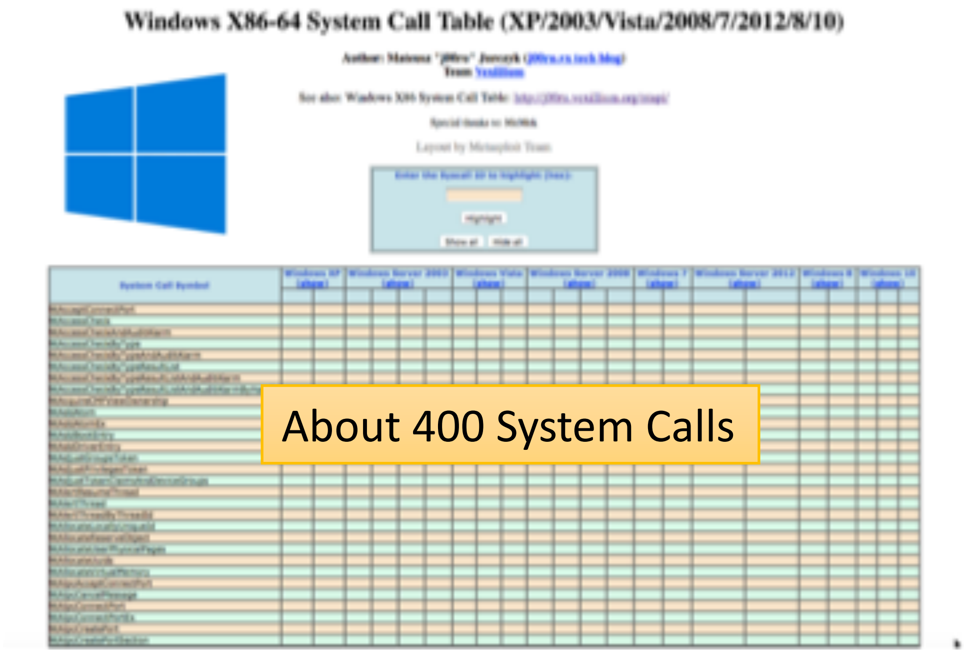
```
#include <unistd.h>
#include <fcntl.h>

#define __NR_open 5
_syscall3(long, open, const char *, filename, int, flags, int, mode)

int main() {
    int filedesc = open("testfile.txt", O_WRONLY | O_APPEND);
    if(filedesc < 0){
        return 1;
    }
    return 0;
}
```

Without GNU C Library (Glibc)

http://stackoverflow.com/questions/11609110/how-to-access-the-system-call-from-user-space

https://github.com/torvalds/linux/blob/master/include/linux/syscalls.h

POSIX API for POSIX-based systems (including virtually all versions of UNIX, Linux, and Mac OS X)



http://j00ru.vexillium.org/ntapi_64/

Win32 API for Windows

Add a System Call to Linux that prints:
"I am awesome!"

References:

- https://www.youtube.com/watch?v=5rr_VoQCOgE
- http://franksthinktank.com/howto/addsyscall/
- https://tssurya.wordpress.com/2014/08/19/adding-a-hello-world-system-call-to-linux-kernel-3-16-0/

Create, Delete Communication Connection
Message Passing Model Host/Process Name
Shared-Memory Model
Transfer Status Information
Attach/Detach Remote Devices

Create/Terminate/Load/Execute Process
Get/Set Process Attributes
Wait for Time/Event
wait event, signal event
Allocate/Free/Dump Memory
Locks for Process Synchronization

Control access to resources
Get and set permissions
Allow and deny user access

Create/Delete/Open/Close/Read/Write File
Get/Set File Attributes

Get/Set Time or Date
Get/Set System Data

Request/Release/Read/Write Device
Get/Set Device Attributes
Logically Attach/Detach devices

Examples of Windows and Unix System Calls

	Windows	Unix
Process Control	CreateProcess() ExitProcess() WaitForSingleObject()	fork() exit() wait()
File Manipulation	CreateFile() ReadFile() WriteFile() CloseHandle()	open() read() write() close()
Device Manipulation	SetConsoleMode() ReadConsole() WriteConsole()	ioctl() read() write()
Information Maintenance	GetCurrentProcessID() SetTimer() Sleep()	getpid() alarm() sleep()
Communication	CreatePipe() CreateFileMapping() MapViewOfFile()	pipe() shmget() mmap()
Protection	SetFileSecurity() InitlializeSecurityDescriptor() SetSecurityDescriptorGroup()	chmod() umask() chown()

System Programs (Utilities)

provide a convenient environment for program development and execution

Some of them are simply user interfaces to system calls; others are considerably more complex

File Manipulation

Create, delete, copy, rename, print, dump, list, and generally manipulate files and directories

Status Information

System Info, Hardware Status, Registry

Background Services (Daemons)

Launch at boot time
Some for system startup, then terminate
Some from system boot to shutdown
Disk checking, process scheduling, error logging
Run in user context not kernel context

Program Loading and Execution

Absolute loaders, relocatable loaders, linkage editors, and overlay-loaders, debugging systems for higher-level and machine language

Programming Language Support

Compilers, assemblers, debuggers and interpreters sometimes provided

Communications

Provide the mechanism for creating virtual connections among processes, users, and computer systems

Application Programs

Run by users
Not typically considered part of OS
Launched by command line, mouse click, finger poke

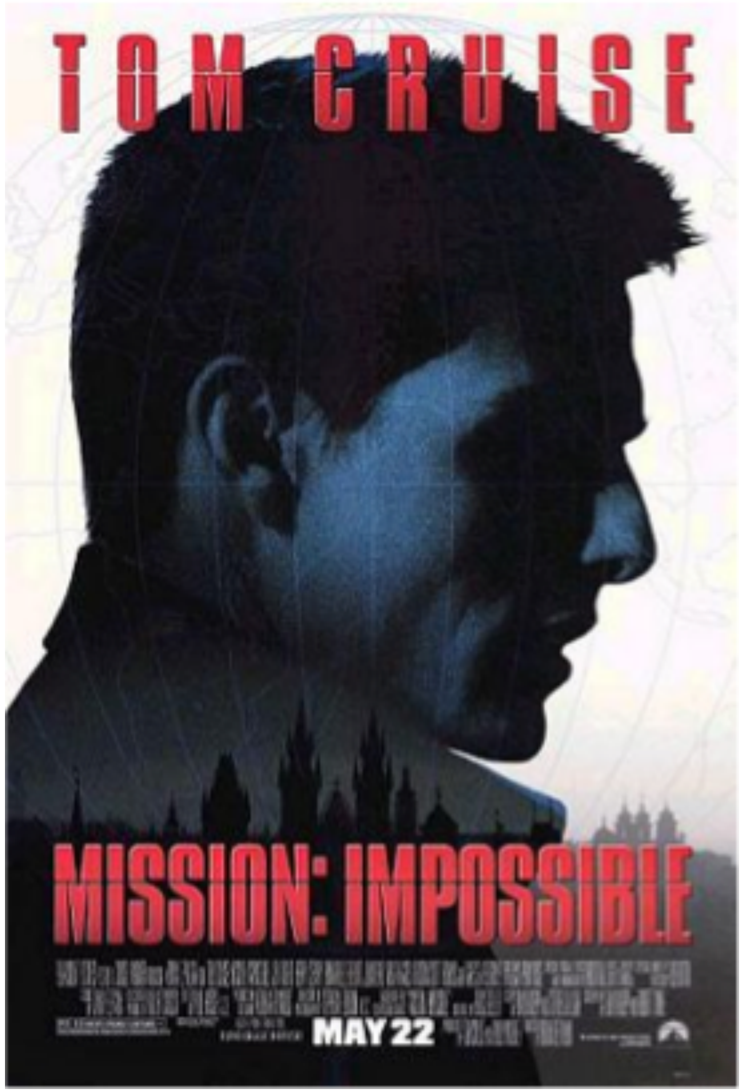
Operating System Design and Implementation

There is no perfect OS, but some have proven to be successful

Define the User/System Goals

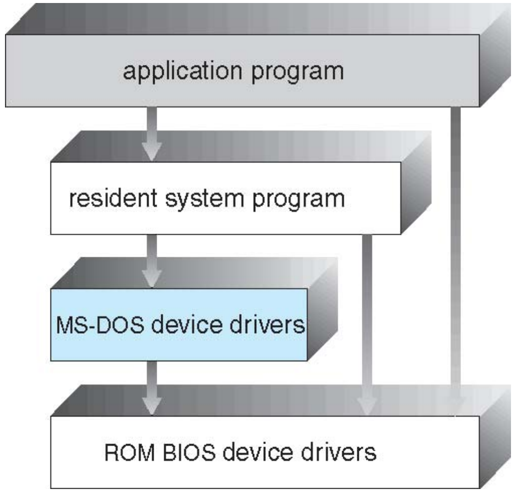
Define the Policies (**What** will be done?) and Mechanisms (**How** to do it?)

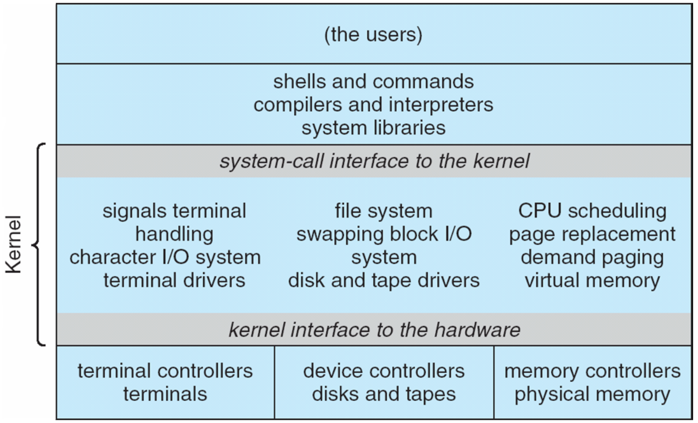
Internal structure of different Operating Systems can vary widely; Affected by choice of hardware, type of system



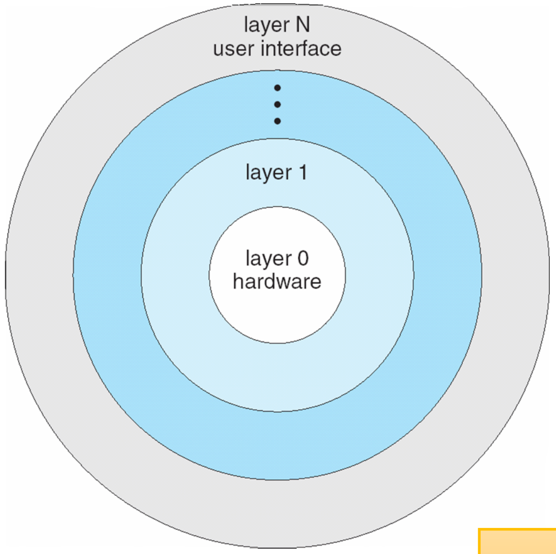
Operating System Structure

Simple Structure

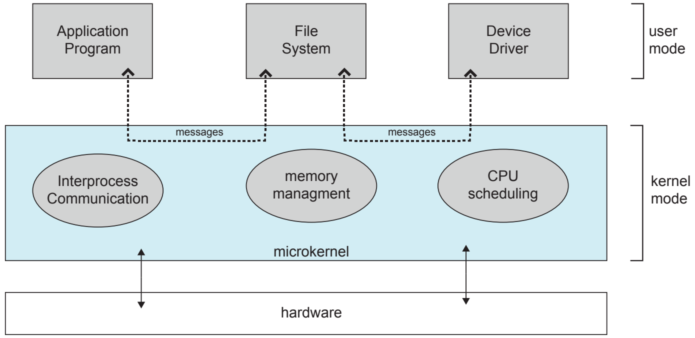




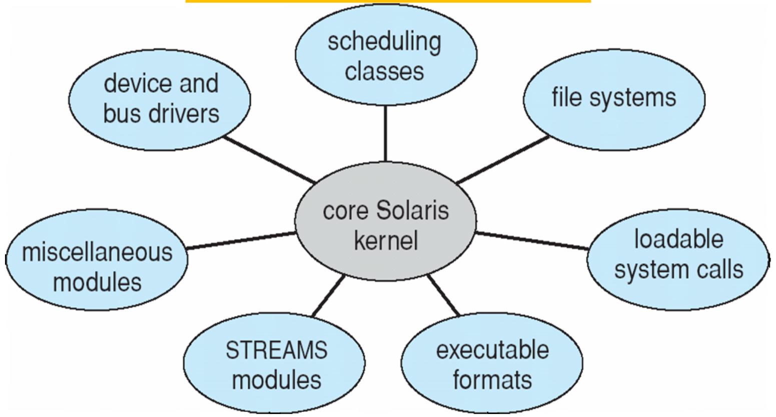
Layered Structure



Microkernel Structure

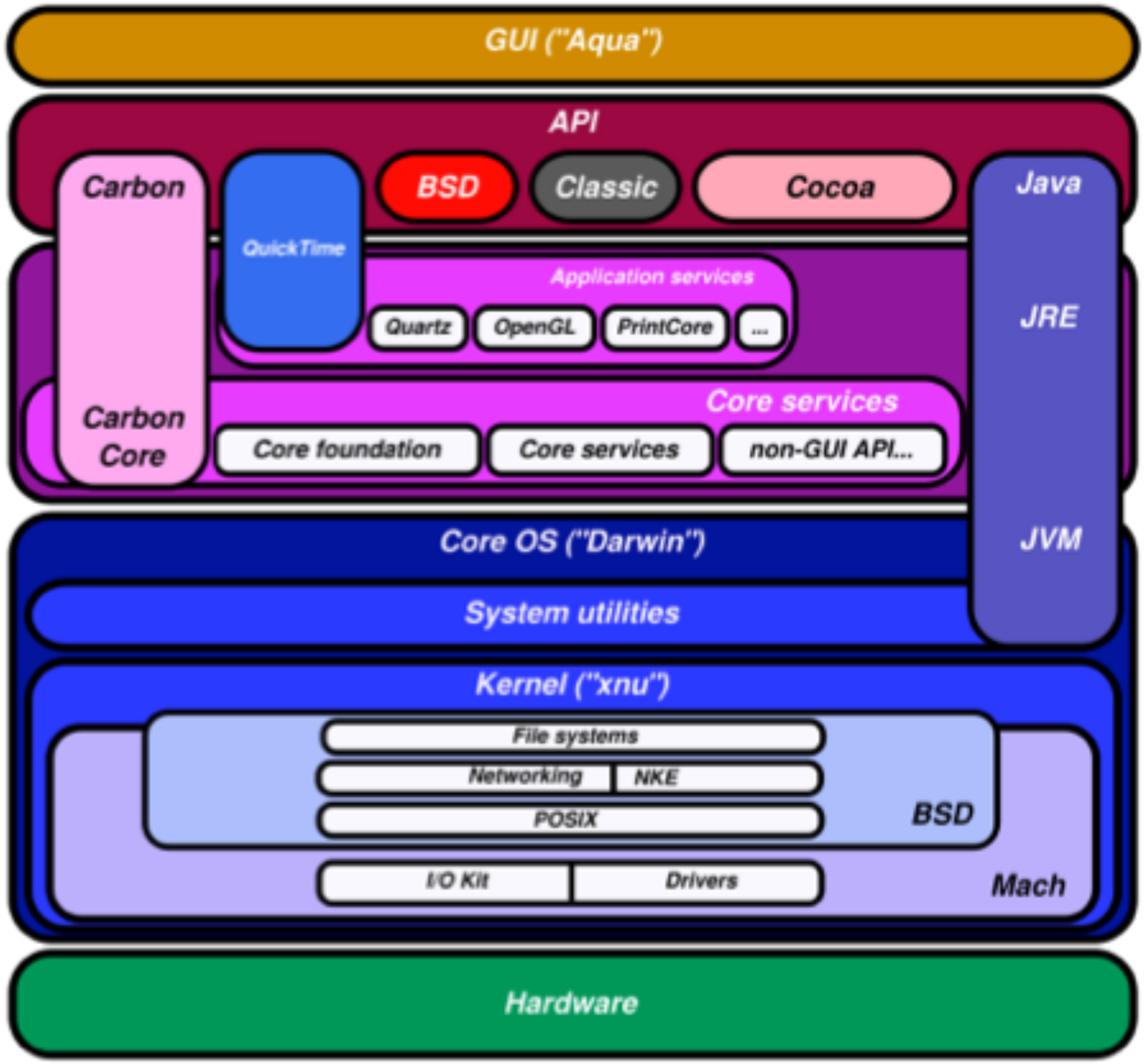


Modular Structure

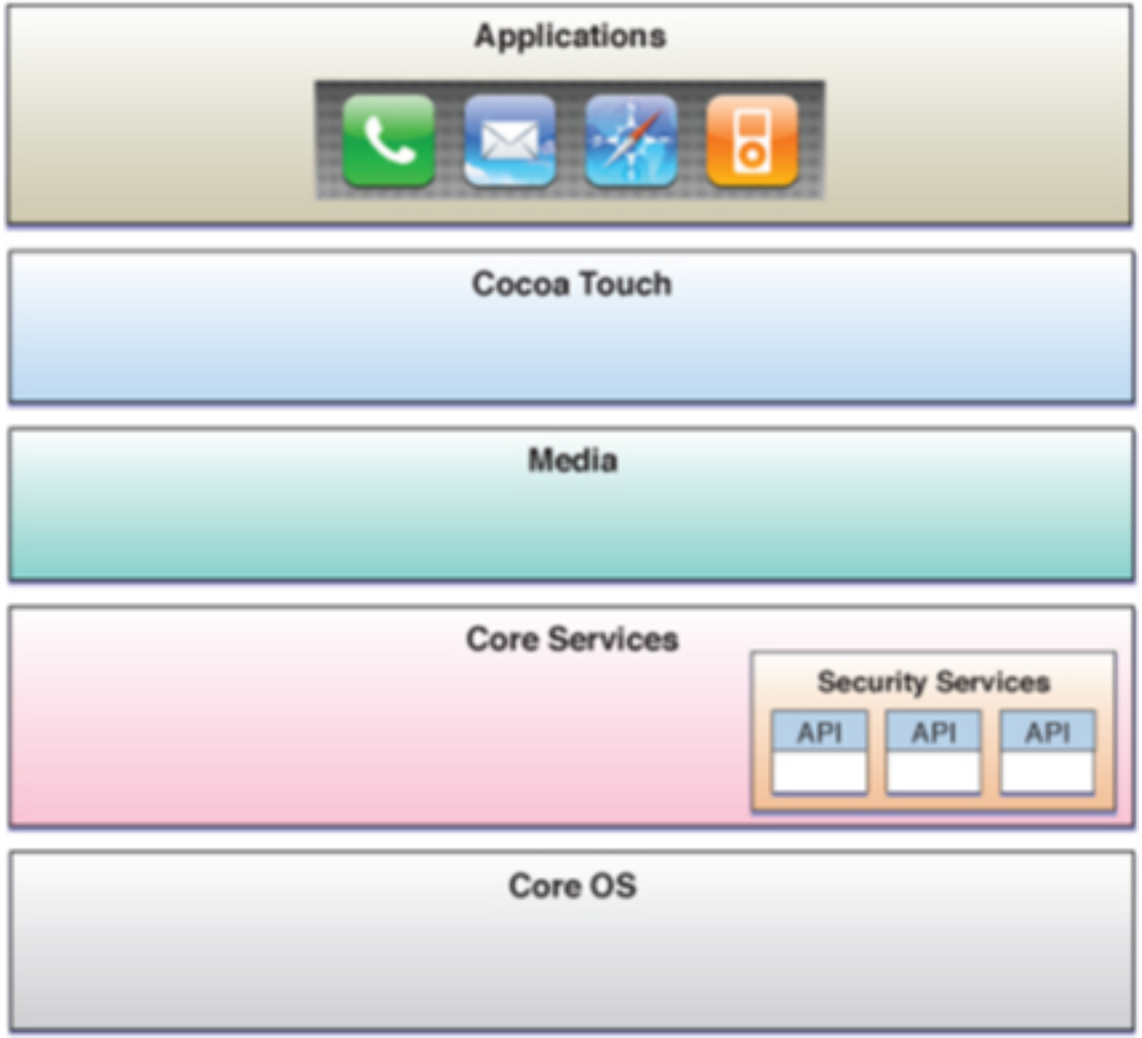


Hybrid Structure

Mac OS Structure



iOS Structure



Android Structure

https://developer.android.com/guide/platform/index.html

Operating System Debugging

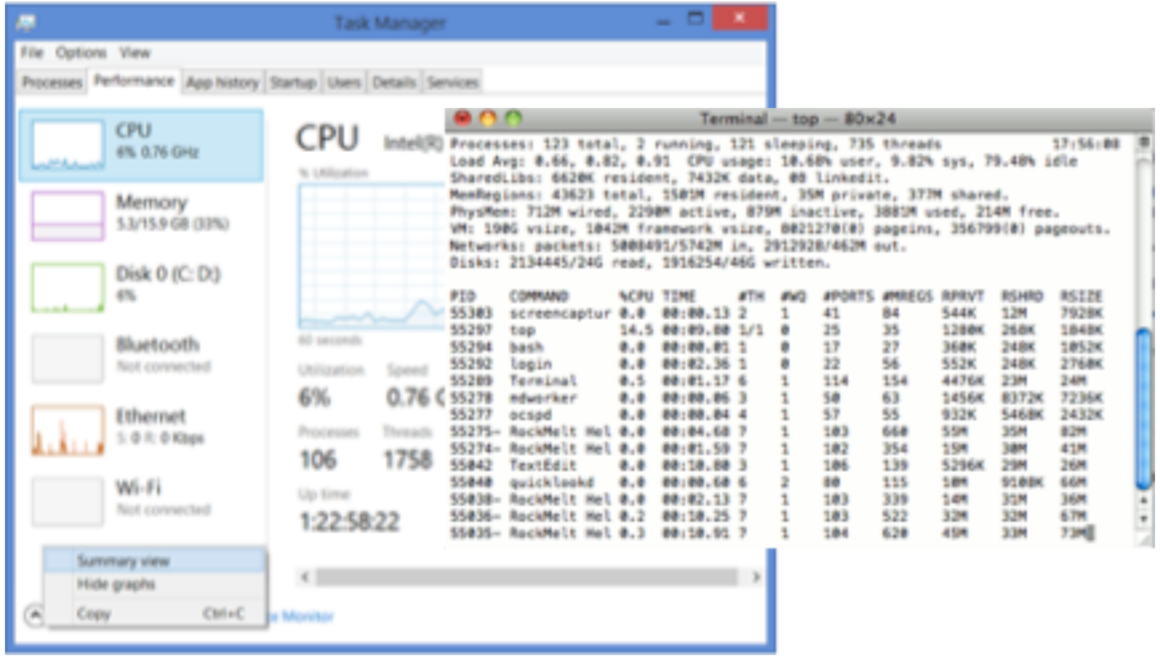
Kernighan's Law: "Debugging is twice as hard as writing the code in the first place. Therefore, if you write the code as cleverly as possible, you are, by definition, not smart enough to debug it."

OS generates *log* files containing error information

Application failure can generate **core dump** file capturing memory of the process

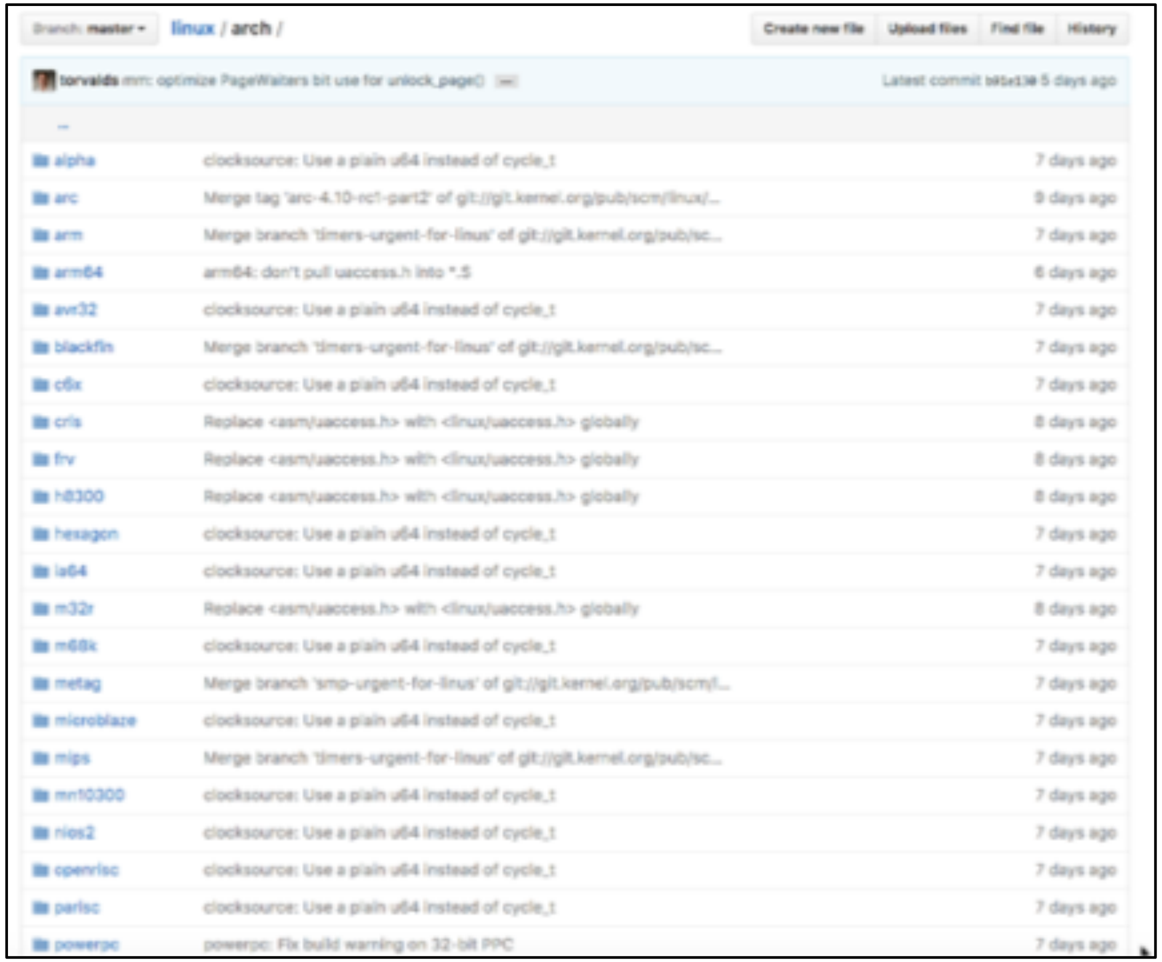
OS failure can generate **crash dump** file containing kernel memory

OS must provide means of computing and displaying measures of system behavior



Operating System Generation

Operating systems are designed to run on any of a class of machines; the system must be configured by obtaining information concerning the specific configuration of the hardware system



https://github.com/torvalds/linux/tree/master/arch

つづく