

CPE 460 Operating System Design

Lecture 2: Operating Systems Structures

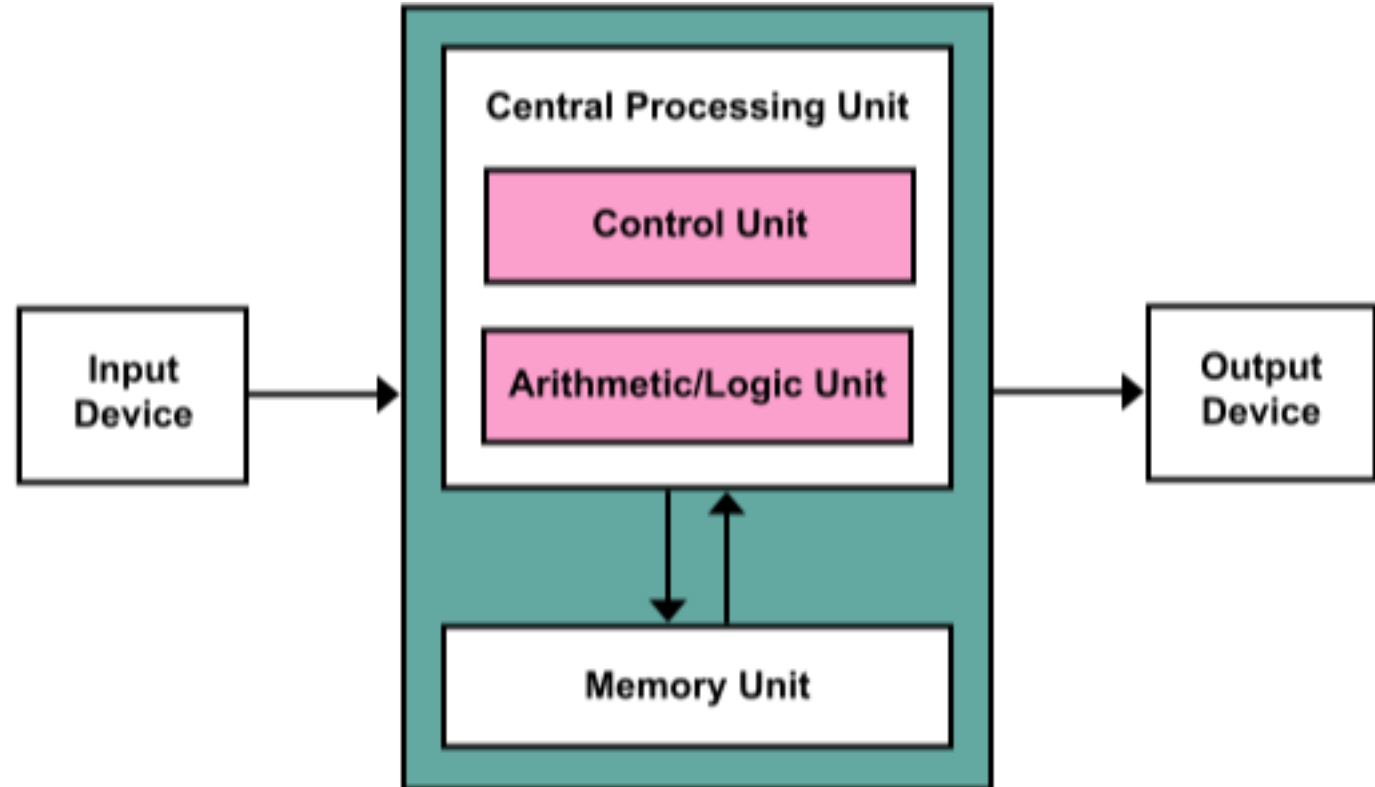
Ahmed Tamrawi

February 13, 2017

1945 Von Neumann Architecture



John von Neumann



Computer System Architecture

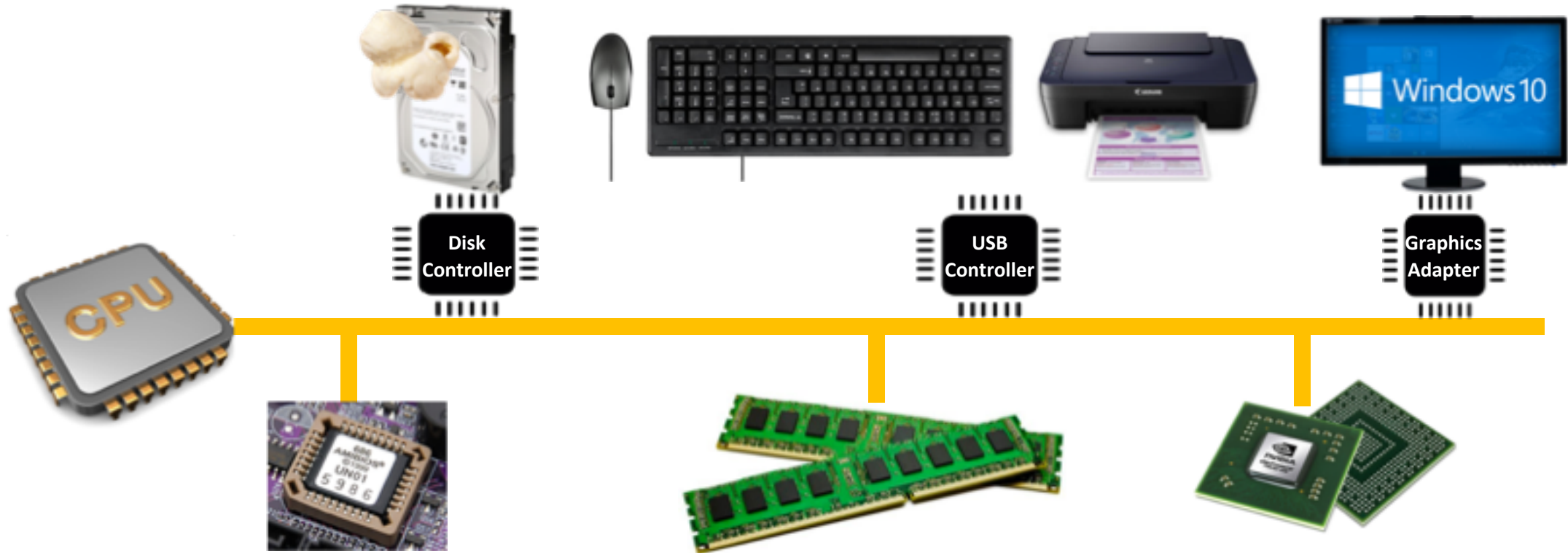
Single Processor System

Clustered System

Multiprocessor System

Single Processor System

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



Multiprocessor System

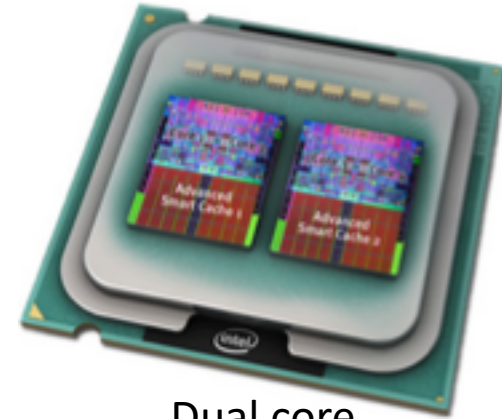
Asymmetric Multiprocessing

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

Symmetric Multiprocessing



Dual Processor



Dual core

Parallelization

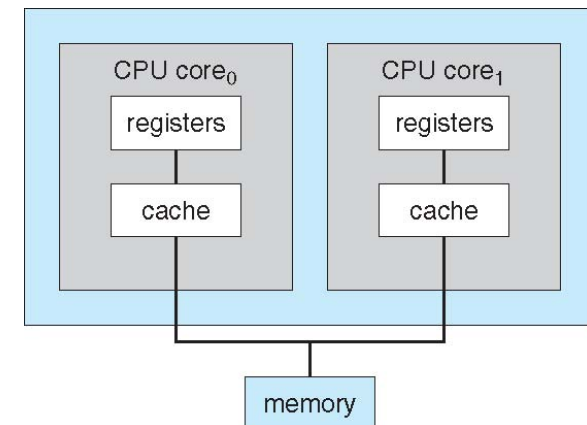
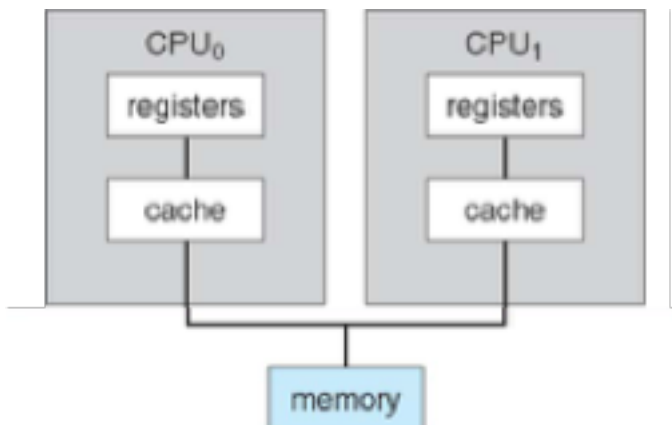
Increased Throughput

Economy of Scale

Increased Reliability

Graceful Degradation

Fault Tolerant



Multiprocessor System

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

**Asymmetric
Multiprocessing**

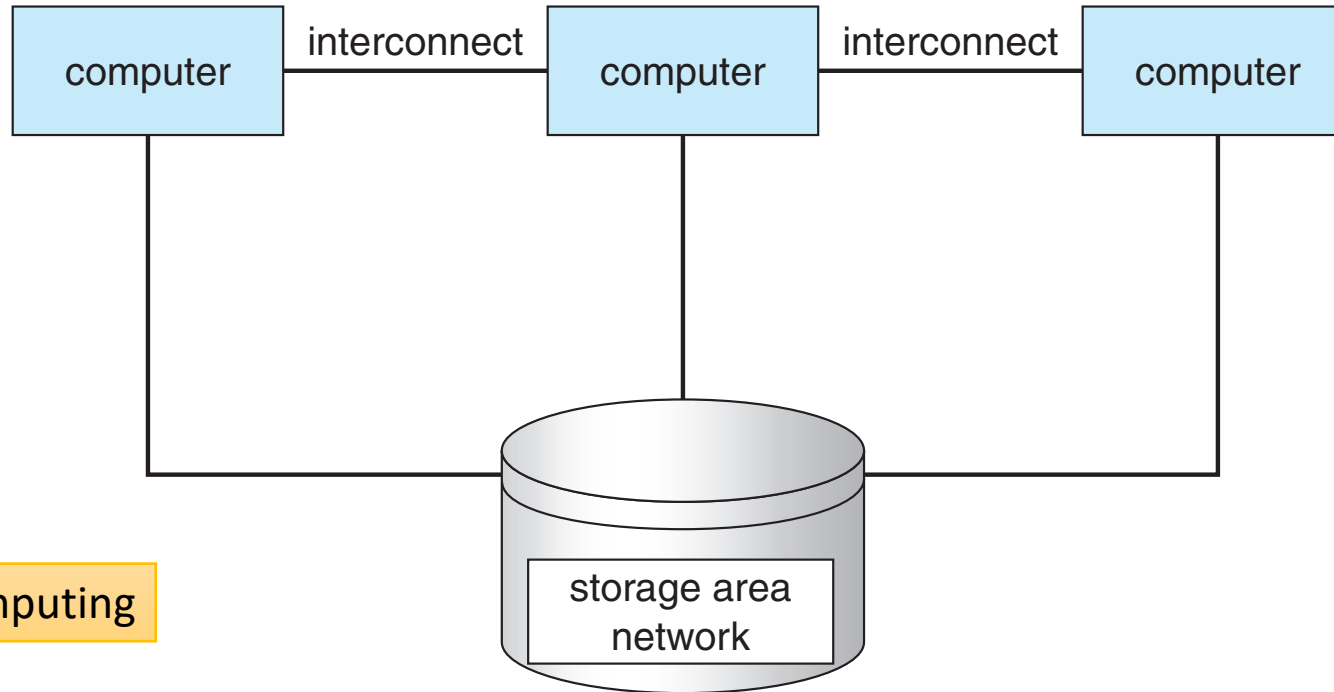
**Symmetric
Multiprocessing**

Clustered System

Multiple systems working together

Asymmetric
Clustering

Symmetric
Clustering



High-Performance Computing

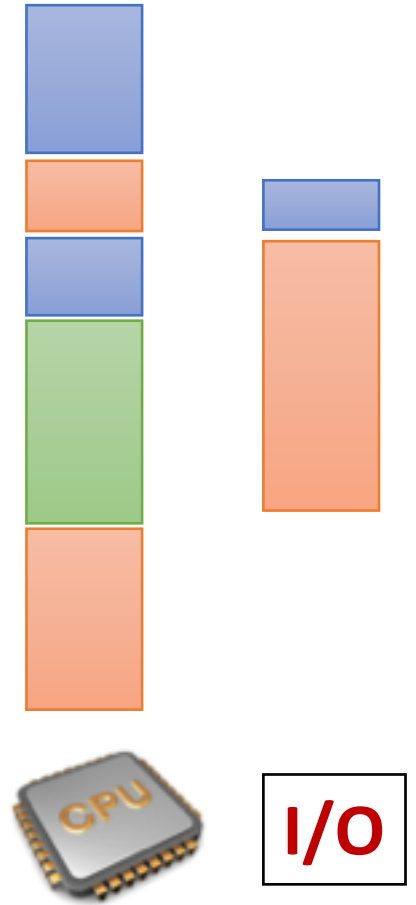
High Availability

Computer System Structure

Multiprogramming (Batch System)

Timesharing (Multitasking)

Multiprogramming (Batch System)

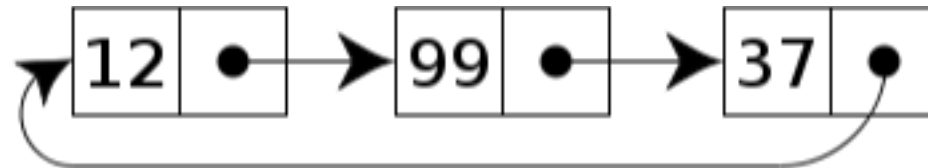
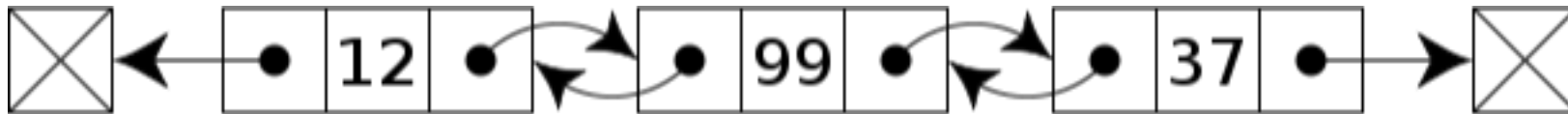
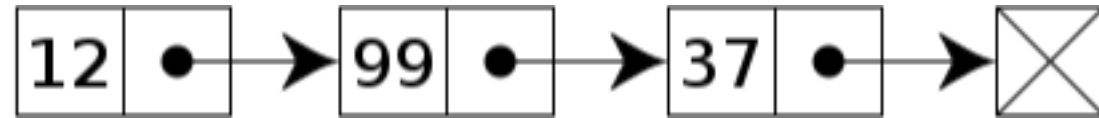


Timesharing (Multitasking)

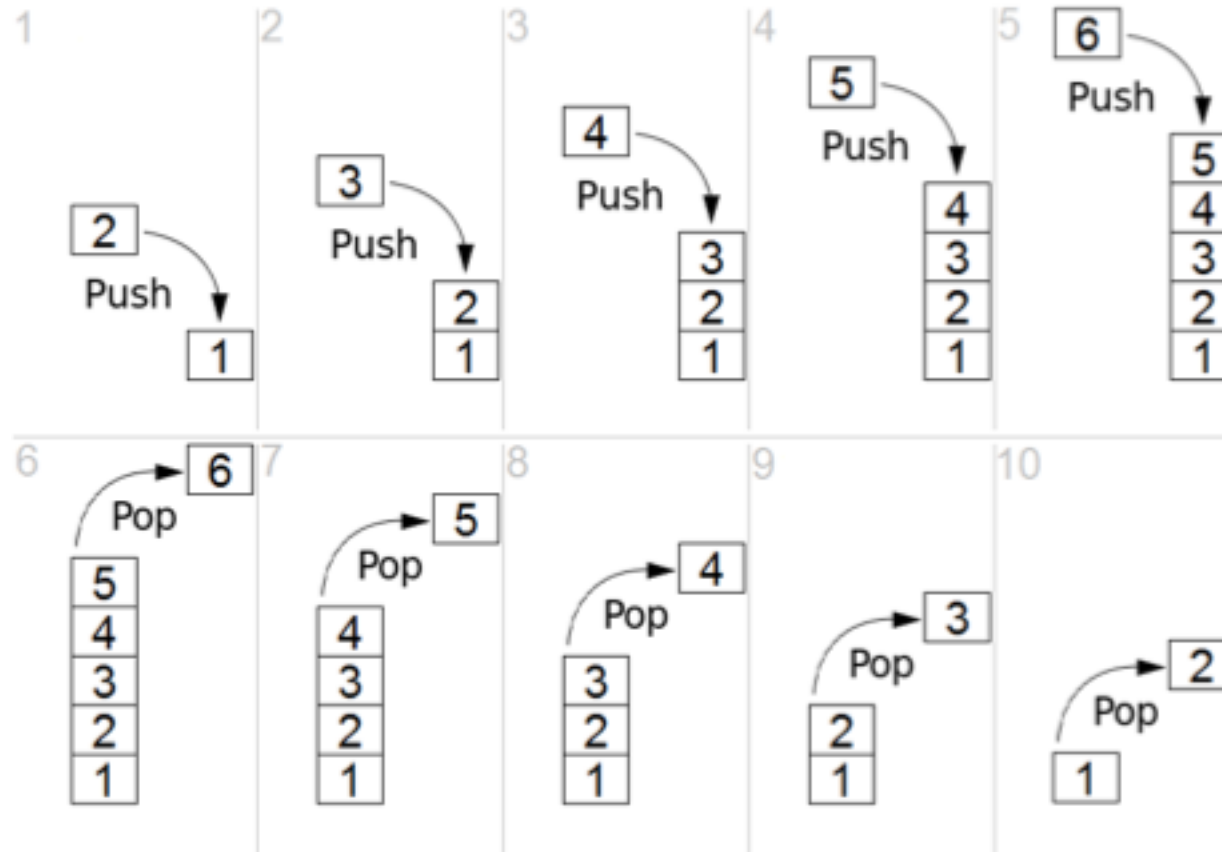


Kernel Data Structures

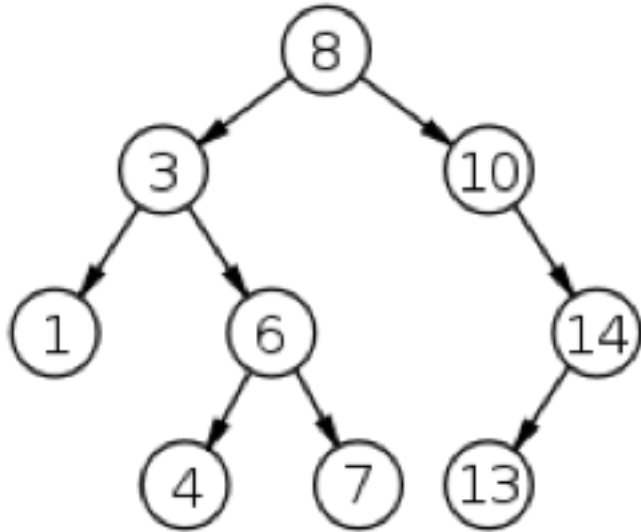
Linked List



Stack

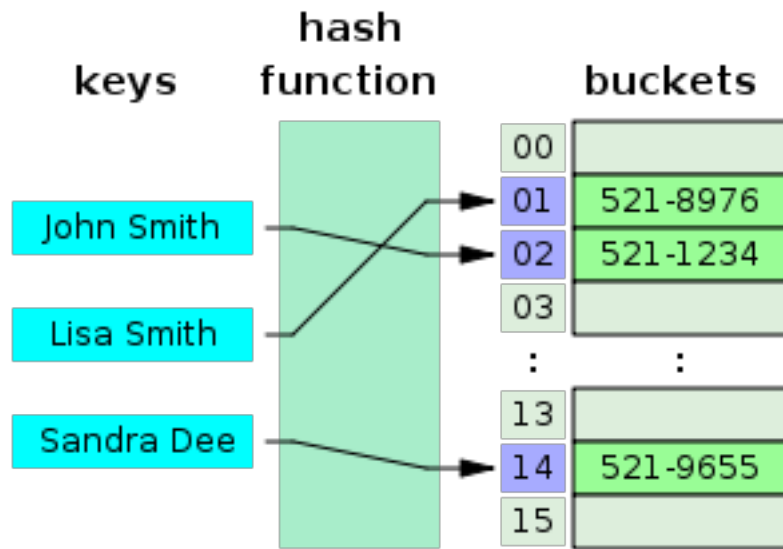


Binary Search Trees



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	$\Theta(\log n)$	$O(n)$

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

101011110001110101



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

Linux Travolds

```

1 #ifndef LINUX_LIST_H
2 #define LINUX_LIST_H
3
4 #include <linux/types.h>
5 #include <linux/spinlock.h>
6 #include <linux/rwlock.h>
7 #include <linux/bit_spinlock.h>
8 #include <linux/errno.h>
9
10 #endif
11
12 /*
13  * Single doubly linked list implementation.
14  *
15  * Nodes of the internal functions ("list") are useful when
16  * representing stack data rather than single entries, as
17  * pointers are already from the recursive entries and so can
18  * generate better code by using these directly rather than
19  * a pointer to the pointer.
20  */
21
22 #define list_entry(ptr, type, member) \
23     ((type *) ((char *) (ptr) - offsetof(type, member)))
24
25 #define list_for_each(ptr, type, member) \
26     for (; ptr; ptr = list_entry(ptr, type, member))
27
28 #define list_for_each_entry(ptr, type, member) \
29     for (ptr = list_entry(ptr, type, member); ptr; ptr = list_entry(ptr, type, member))
30
31 #define list_for_each_entry_safe(ptr, type, member, next) \
32     for (next = list_entry(ptr, type, member); ptr; ptr = next)
33
34 #define list_for_each_reverse(ptr, type, member) \
35     for (ptr; ptr; ptr = list_entry(ptr, type, member))
36
37 #define list_for_each_reverse_entry(ptr, type, member) \
38     for (ptr = list_entry(ptr, type, member); ptr; ptr = list_entry(ptr, type, member))
39
40 #define list_for_each_entry_reverse(ptr, type, member) \
41     for (ptr = list_entry(ptr, type, member); ptr; ptr = list_entry(ptr, type, member))
42
43 #define list_for_each_safe(ptr, type, member, next) \
44     for (next = list_entry(ptr, type, member); ptr; ptr = next)
45
46 #define list_for_each_entry_safe(ptr, type, member, next) \
47     for (next = list_entry(ptr, type, member); ptr; ptr = next)
48
49 #endif
50

```

Doubly Linked List Implementation

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

```

1 /*
2  * A generic kernel FIFO implementation.
3  * Copyright (C) 2003 Richard Knizer <richard@nvidia.com>
4  *
5  * This program is free software; you can redistribute it and/or modify
6  * it under the terms of the GNU General Public License as published by
7  * the Free Software Foundation, either version 2 of the License, or
8  * (at your option) any later version.
9  *
10  * This program is distributed in the hope that it will be useful,
11  * but WITHOUT ANY WARRANTY; without even the implied warranty of
12  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
13  * GNU General Public License for more details.
14  *
15  * You should have received a copy of the GNU General Public License
16  * along with this program; if not, write to the Free Software
17  * Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA
18  */
19
20 #ifndef LINUX_KFIFO_H
21 #define LINUX_KFIFO_H
22
23 #include <linux/types.h>
24 #include <linux/spinlock.h>
25 #include <linux/rwlock.h>
26 #include <linux/bit_spinlock.h>
27 #include <linux/errno.h>
28 #include <linux/string.h>
29
30 #endif
31
32 #define kfifo_entry(ptr, type, member) \
33     ((type *) ((char *) (ptr) - offsetof(type, member)))
34
35 #define kfifo_for_each(ptr, type, member) \
36     for (; ptr; ptr = kfifo_entry(ptr, type, member))
37
38 #define kfifo_for_each_entry(ptr, type, member) \
39     for (ptr = kfifo_entry(ptr, type, member); ptr; ptr = kfifo_entry(ptr, type, member))
40
41 #define kfifo_for_each_entry_safe(ptr, type, member, next) \
42     for (next = kfifo_entry(ptr, type, member); ptr; ptr = next)
43
44 #define kfifo_for_each_reverse(ptr, type, member) \
45     for (ptr; ptr; ptr = kfifo_entry(ptr, type, member))
46
47 #define kfifo_for_each_reverse_entry(ptr, type, member) \
48     for (ptr = kfifo_entry(ptr, type, member); ptr; ptr = kfifo_entry(ptr, type, member))
49
50 #define kfifo_for_each_entry_reverse(ptr, type, member) \
51     for (ptr = kfifo_entry(ptr, type, member); ptr; ptr = kfifo_entry(ptr, type, member))
52
53 #define kfifo_for_each_safe(ptr, type, member, next) \
54     for (next = kfifo_entry(ptr, type, member); ptr; ptr = next)
55
56 #define kfifo_for_each_entry_safe(ptr, type, member, next) \
57     for (next = kfifo_entry(ptr, type, member); ptr; ptr = next)
58
59 #endif
60

```

FIFO Queue Implementation

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

```

1 /*
2  * Red Black Trees implementation.
3  * Copyright (C) 2002 Richard Knizer <richard@nvidia.com>
4  *
5  * This program is free software; you can redistribute it and/or modify
6  * it under the terms of the GNU General Public License as published by
7  * the Free Software Foundation, either version 2 of the License, or
8  * (at your option) any later version.
9  *
10  * This program is distributed in the hope that it will be useful,
11  * but WITHOUT ANY WARRANTY; without even the implied warranty of
12  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
13  * GNU General Public License for more details.
14  *
15  * You should have received a copy of the GNU General Public License
16  * along with this program; if not, write to the Free Software
17  * Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA
18  */
19
20 #ifndef LINUX_RBTREE_H
21 #define LINUX_RBTREE_H
22
23 #include <linux/types.h>
24 #include <linux/spinlock.h>
25 #include <linux/rwlock.h>
26 #include <linux/bit_spinlock.h>
27 #include <linux/errno.h>
28 #include <linux/string.h>
29
30 #endif
31
32 #define rbt_entry(ptr, type, member) \
33     ((type *) ((char *) (ptr) - offsetof(type, member)))
34
35 #define rbt_for_each(ptr, type, member) \
36     for (; ptr; ptr = rbt_entry(ptr, type, member))
37
38 #define rbt_for_each_entry(ptr, type, member) \
39     for (ptr = rbt_entry(ptr, type, member); ptr; ptr = rbt_entry(ptr, type, member))
40
41 #define rbt_for_each_entry_safe(ptr, type, member, next) \
42     for (next = rbt_entry(ptr, type, member); ptr; ptr = next)
43
44 #define rbt_for_each_reverse(ptr, type, member) \
45     for (ptr; ptr; ptr = rbt_entry(ptr, type, member))
46
47 #define rbt_for_each_reverse_entry(ptr, type, member) \
48     for (ptr = rbt_entry(ptr, type, member); ptr; ptr = rbt_entry(ptr, type, member))
49
50 #define rbt_for_each_entry_reverse(ptr, type, member) \
51     for (ptr = rbt_entry(ptr, type, member); ptr; ptr = rbt_entry(ptr, type, member))
52
53 #define rbt_for_each_safe(ptr, type, member, next) \
54     for (next = rbt_entry(ptr, type, member); ptr; ptr = next)
55
56 #define rbt_for_each_entry_safe(ptr, type, member, next) \
57     for (next = rbt_entry(ptr, type, member); ptr; ptr = next)
58
59 #endif
60

```

Red Black Trees Implementation

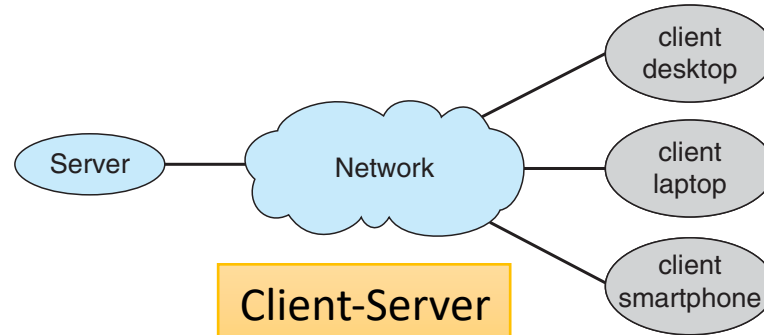
<https://github.com/torvalds/linux/blob/master/include/linux/rbtree.h>



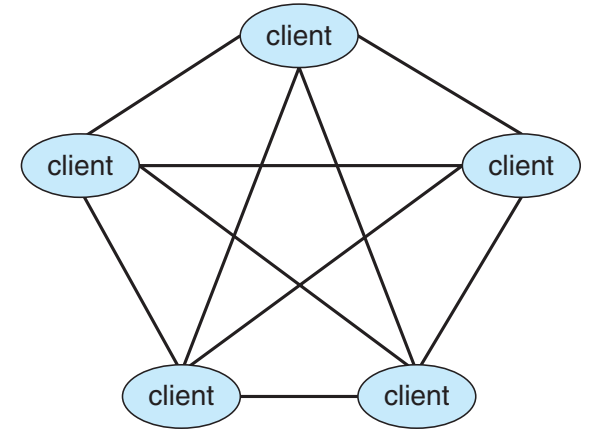
Computing Environments



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



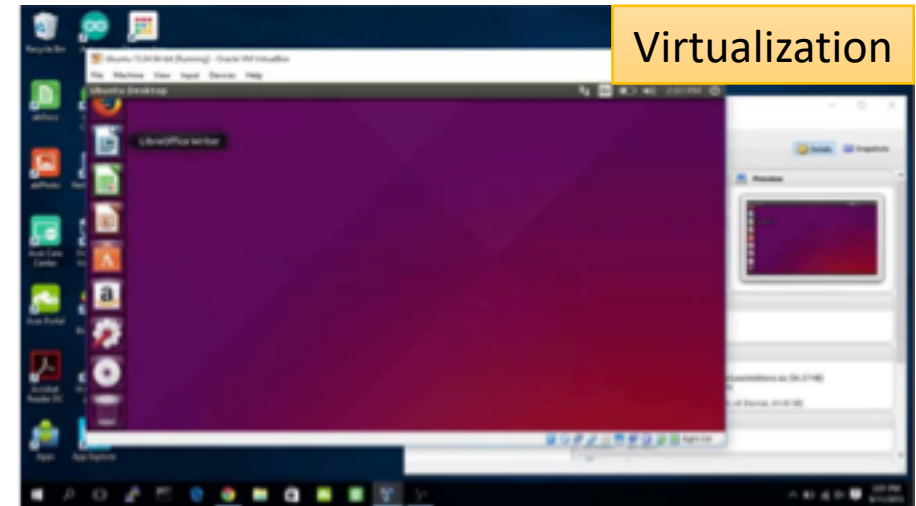
Motorola 68k CPU

Emulation

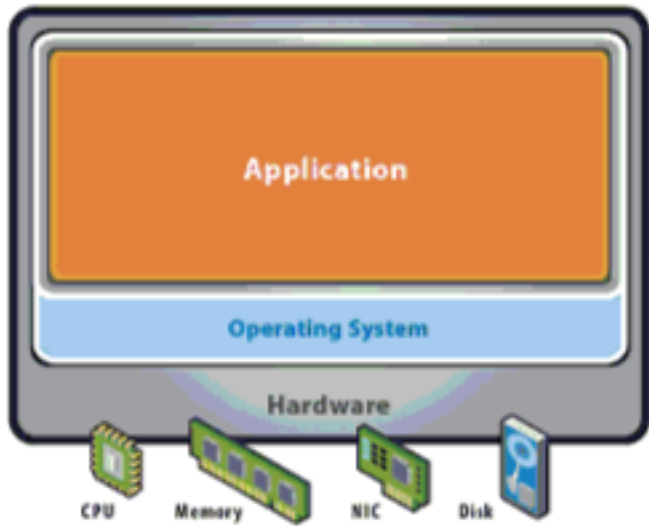


Intel x86 CPU

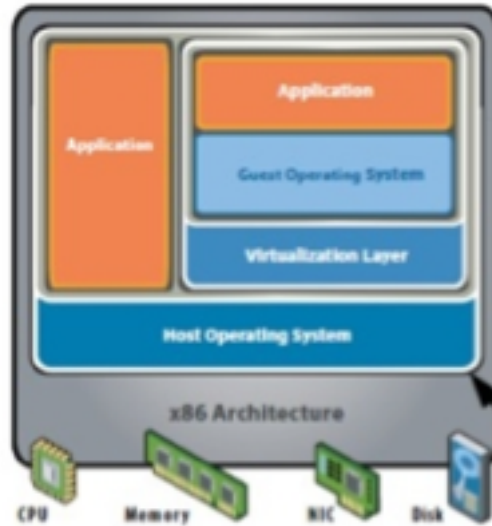
Emulation used when source CPU type different from target type



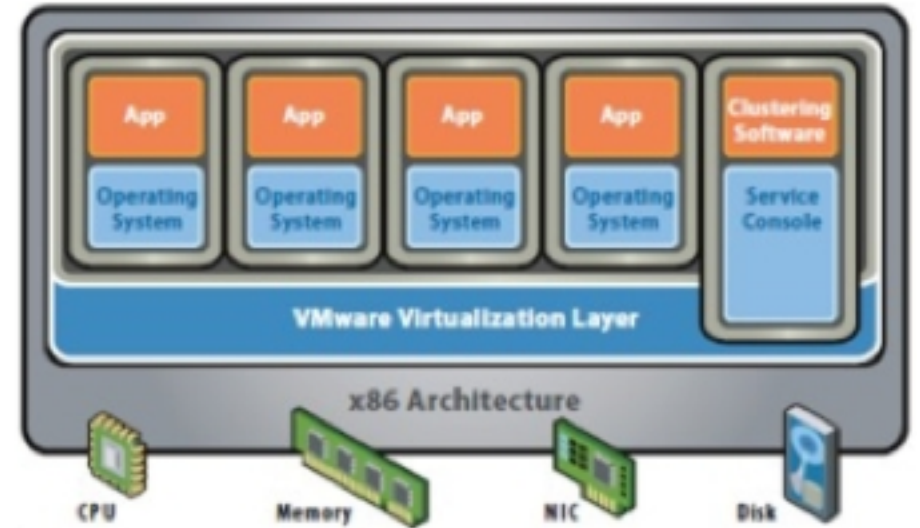
OS natively compiled for CPU, running guest OSes also natively compiled



No Virtualization



OS-Level Virtualization



Full Virtualization



open source



<https://github.com/torvalds/linux>



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



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

freeBSD®

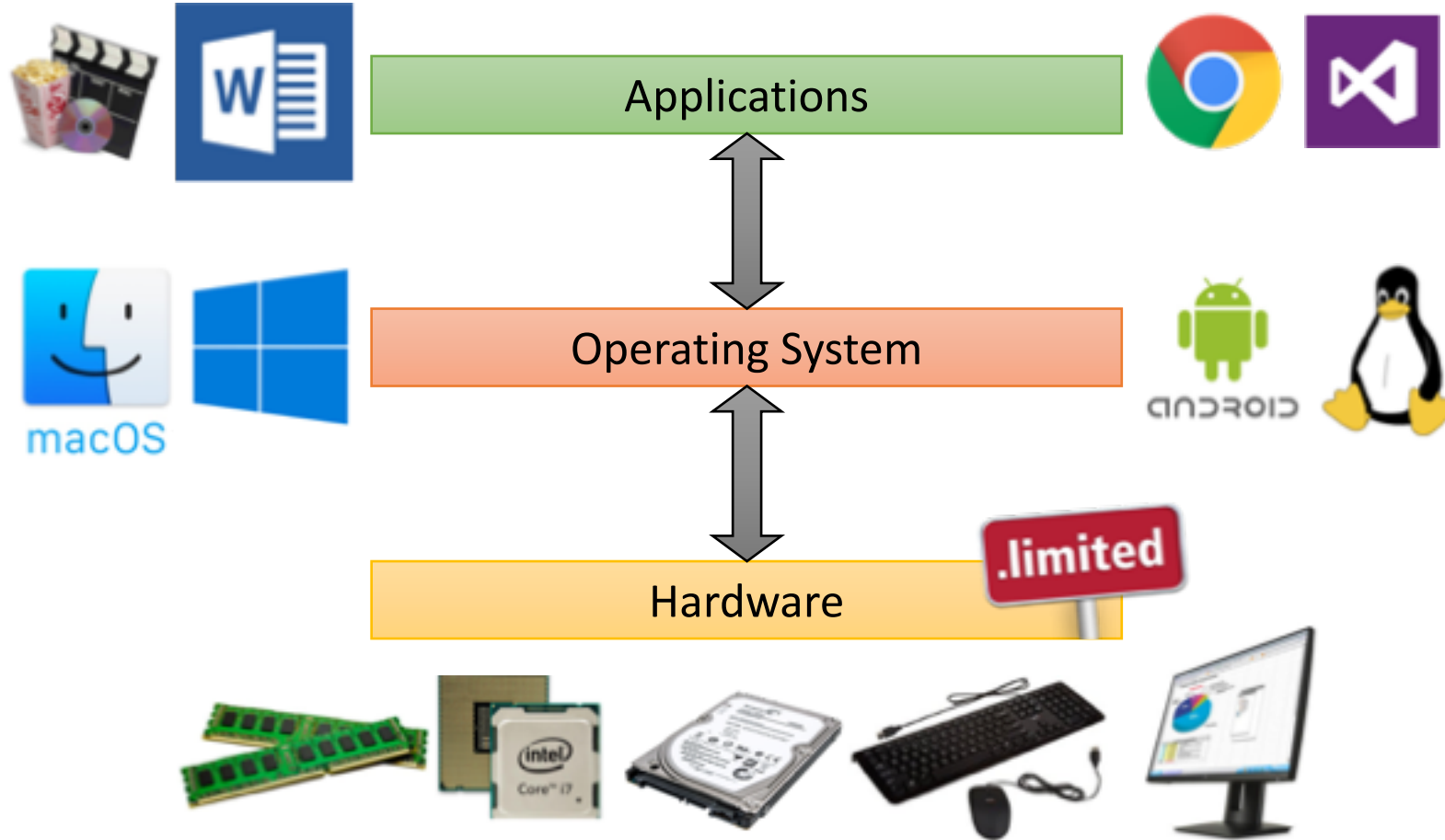


<https://github.com/android>

Realistic View of Operating System

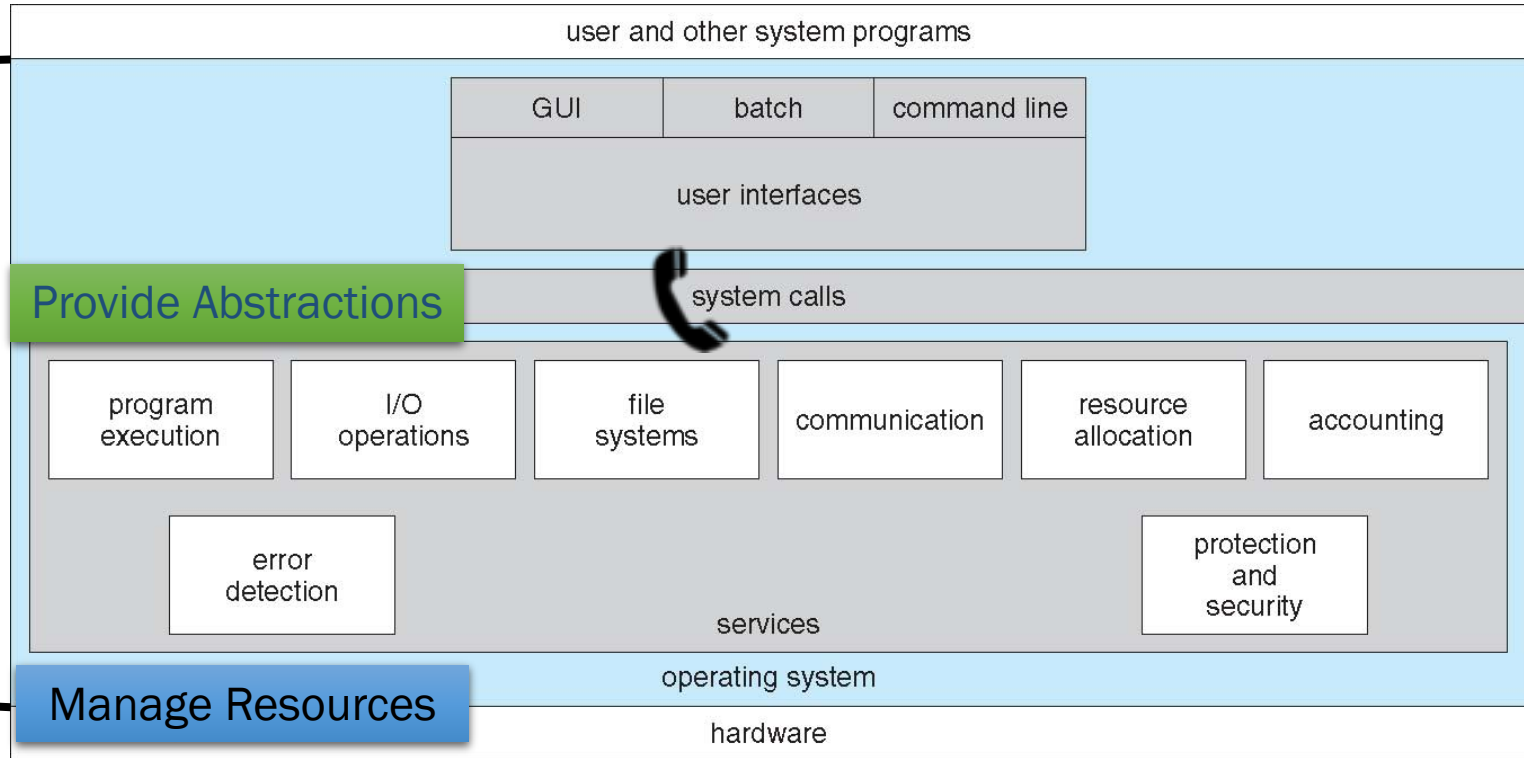
Manage Resources

Provide Abstractions



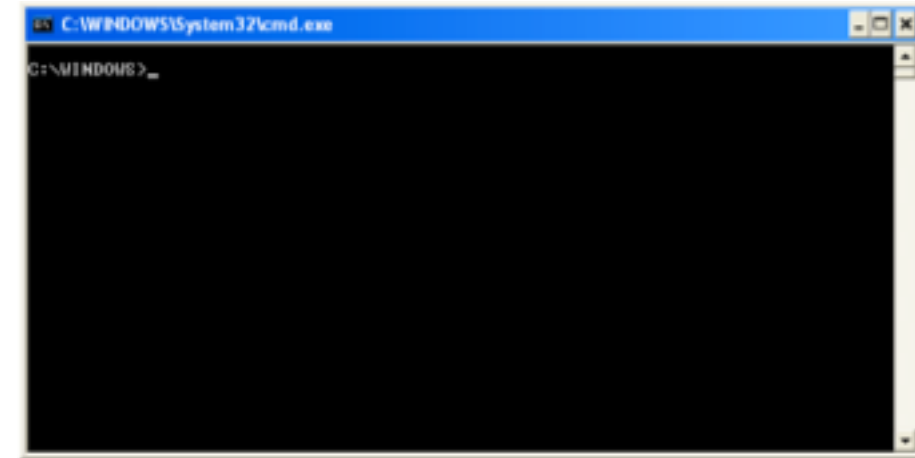
A View of Operating System Services

*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



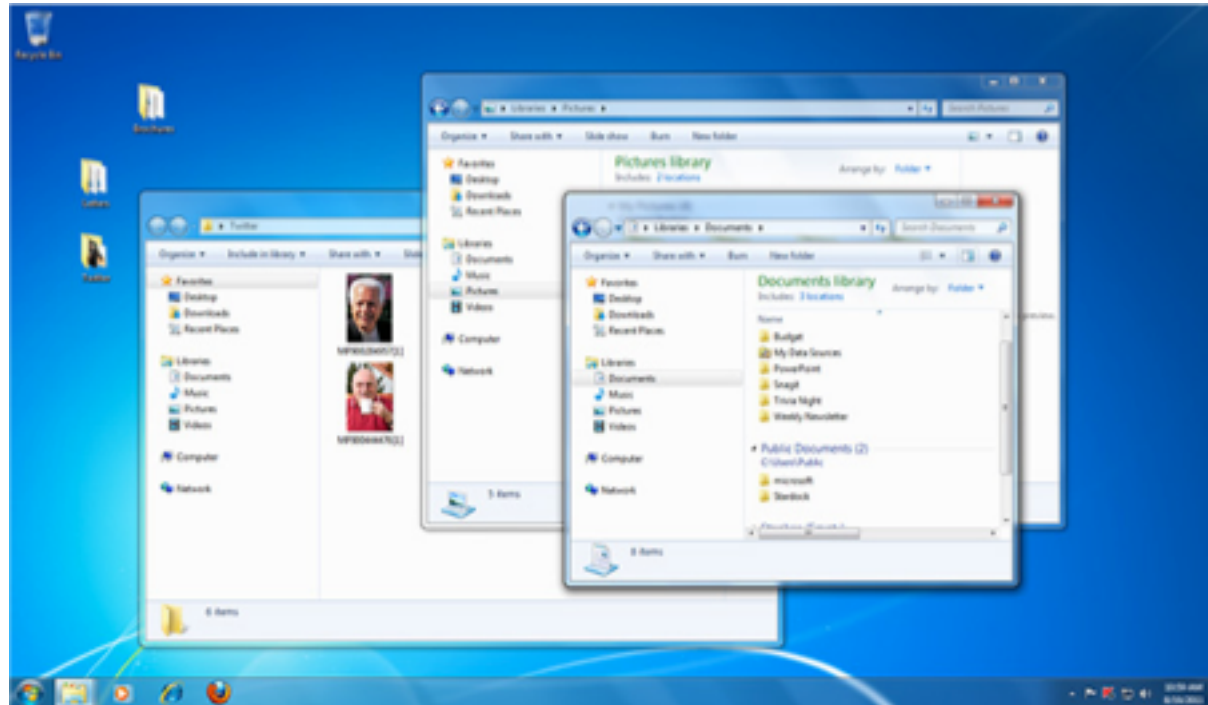
Sometimes implemented in kernel, sometimes by systems program

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



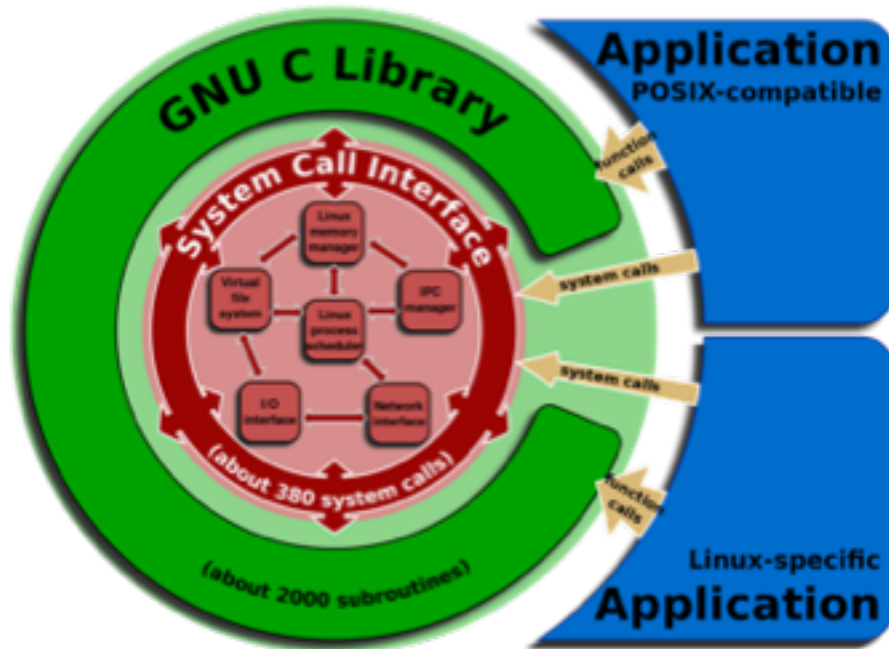


Software Interrupt (Trap)

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

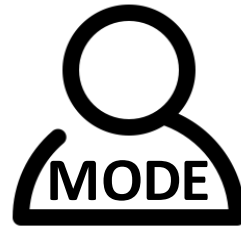
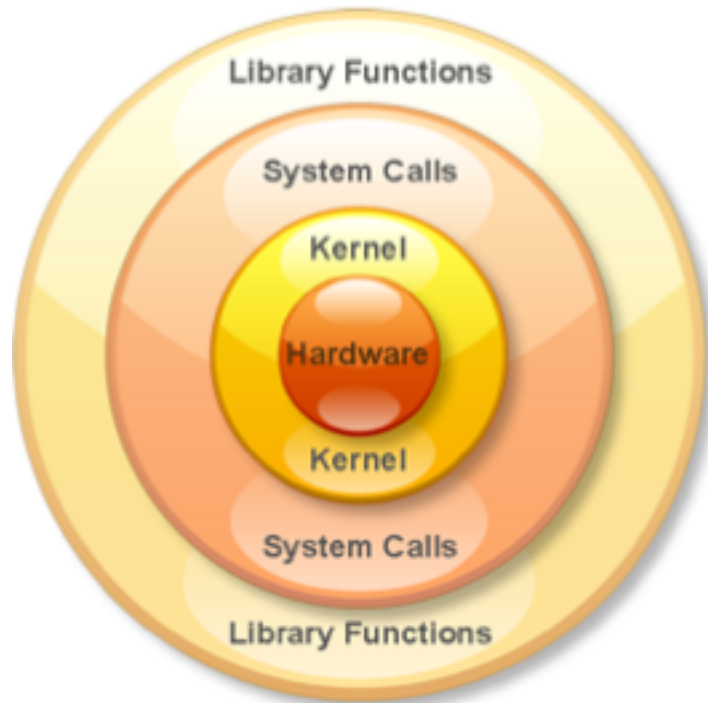


Get/Set Time or Date
 Get/Set System Data



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

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

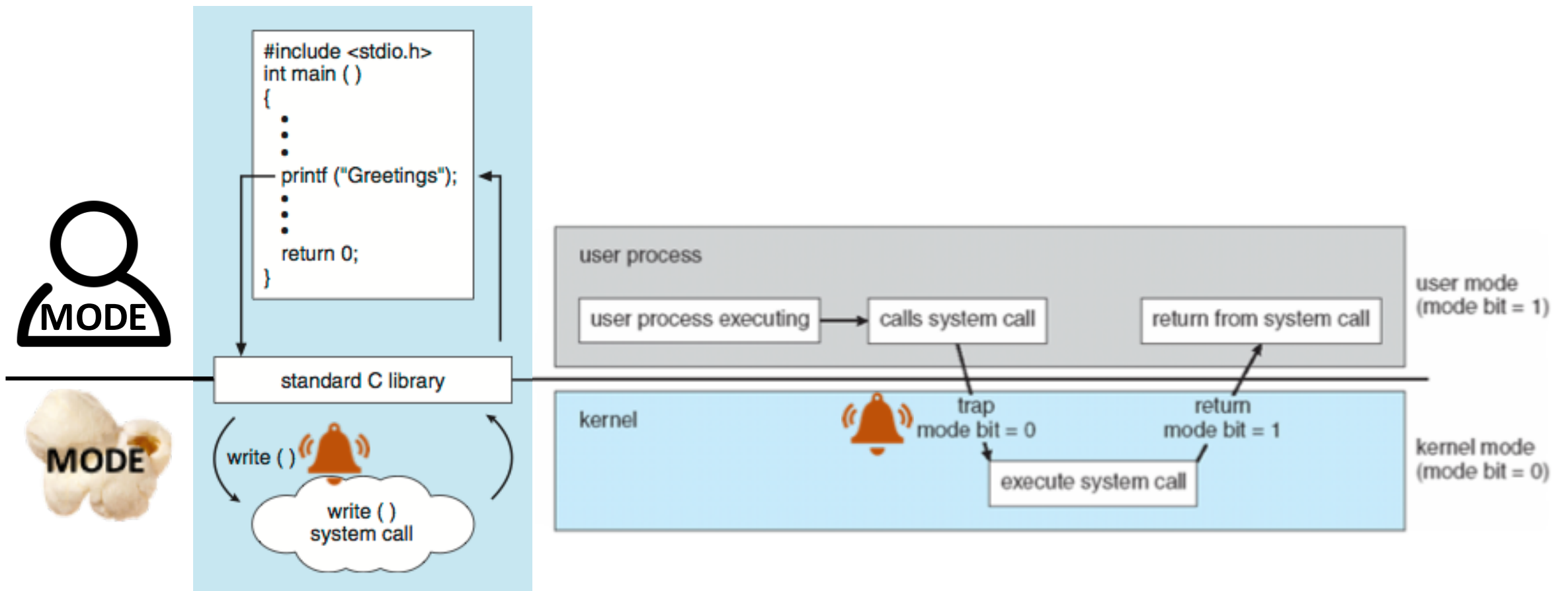



```
ls --color=tty (zsh)
tratum → Test ls
TestFile.txt
tratum → Test cp TestFile.txt TestFile-Copy.txt
tratum → Test ls
TestFile-Copy.txt TestFile.txt
tratum → Test
```

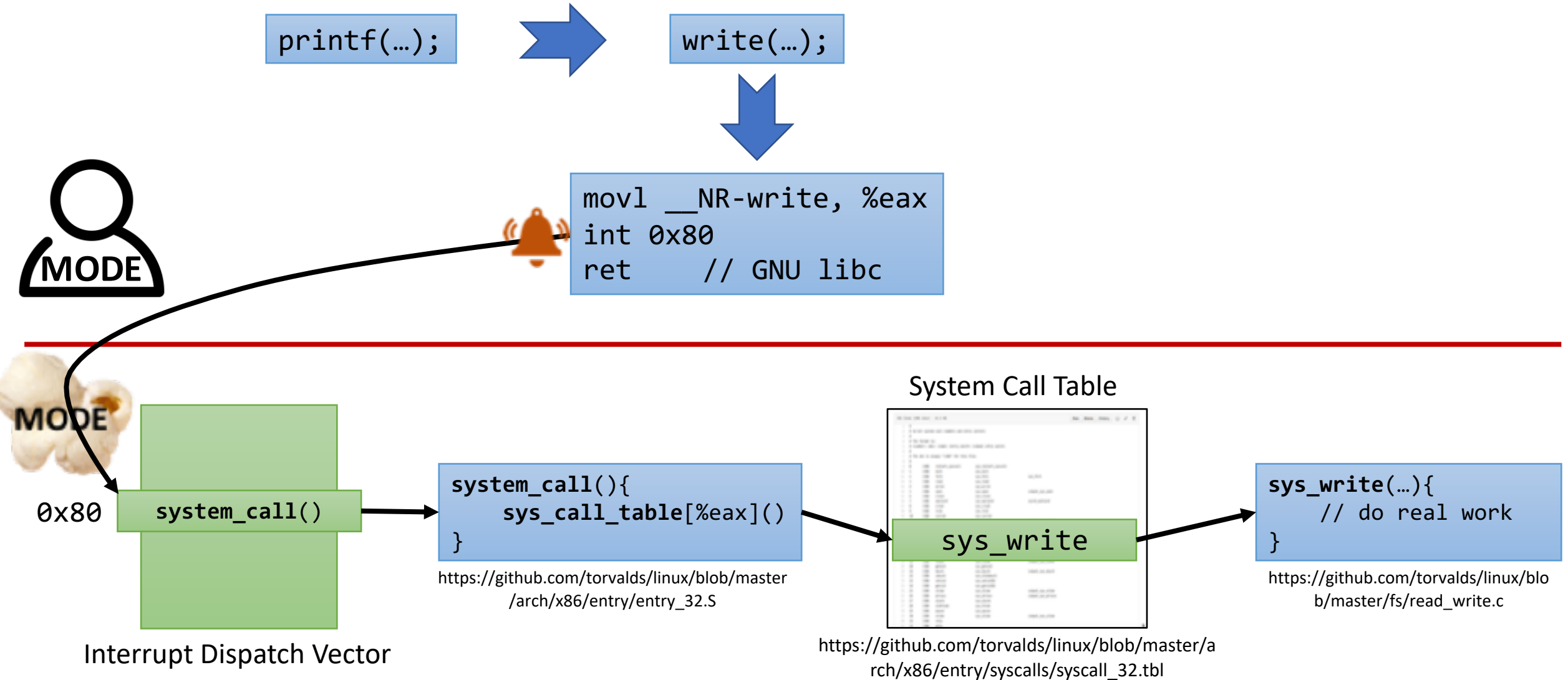
```
strace cp TestFile.txt TestFile-Copy.txt
```

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



How the kernel know which system call to execute?



How Parameters are Passed?

Registers

More parameters than registers

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



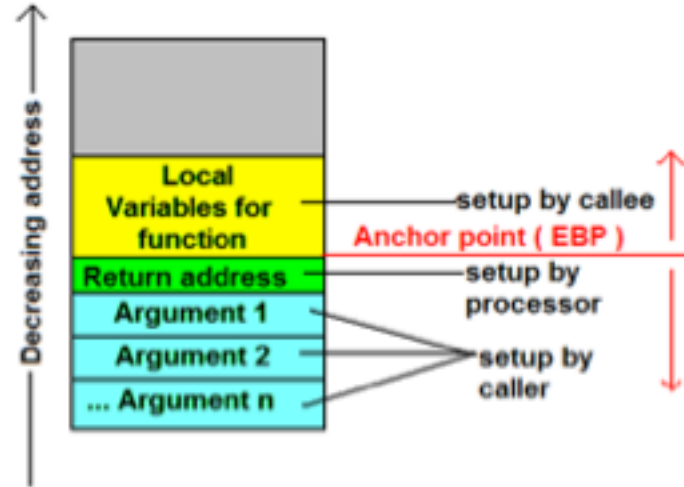
```
write(screen, "Hello!", ...);
```



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

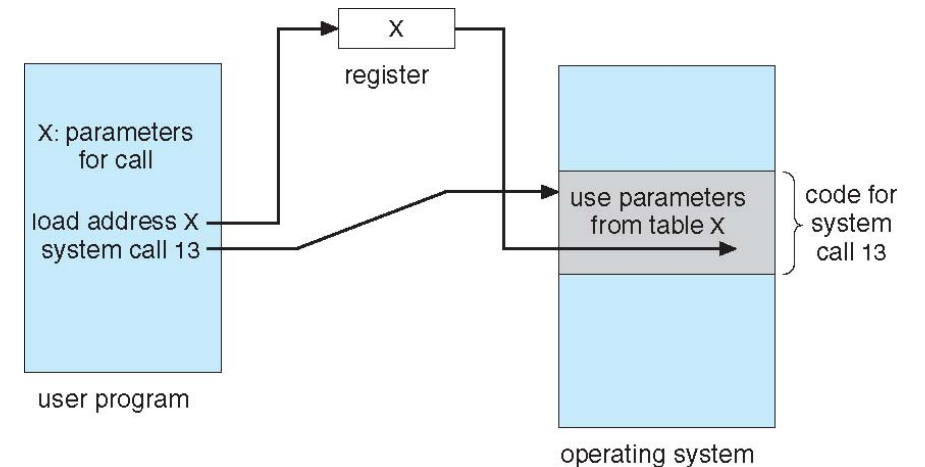
User-Mode Stack

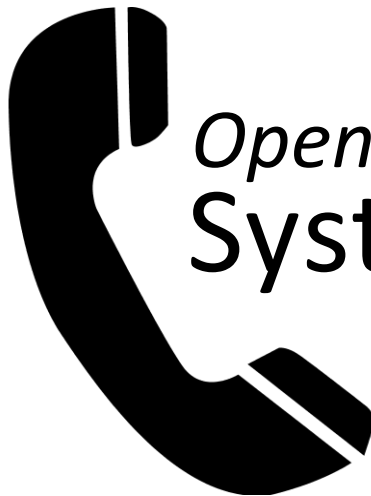
Do not limit the number or length of parameters being passed



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

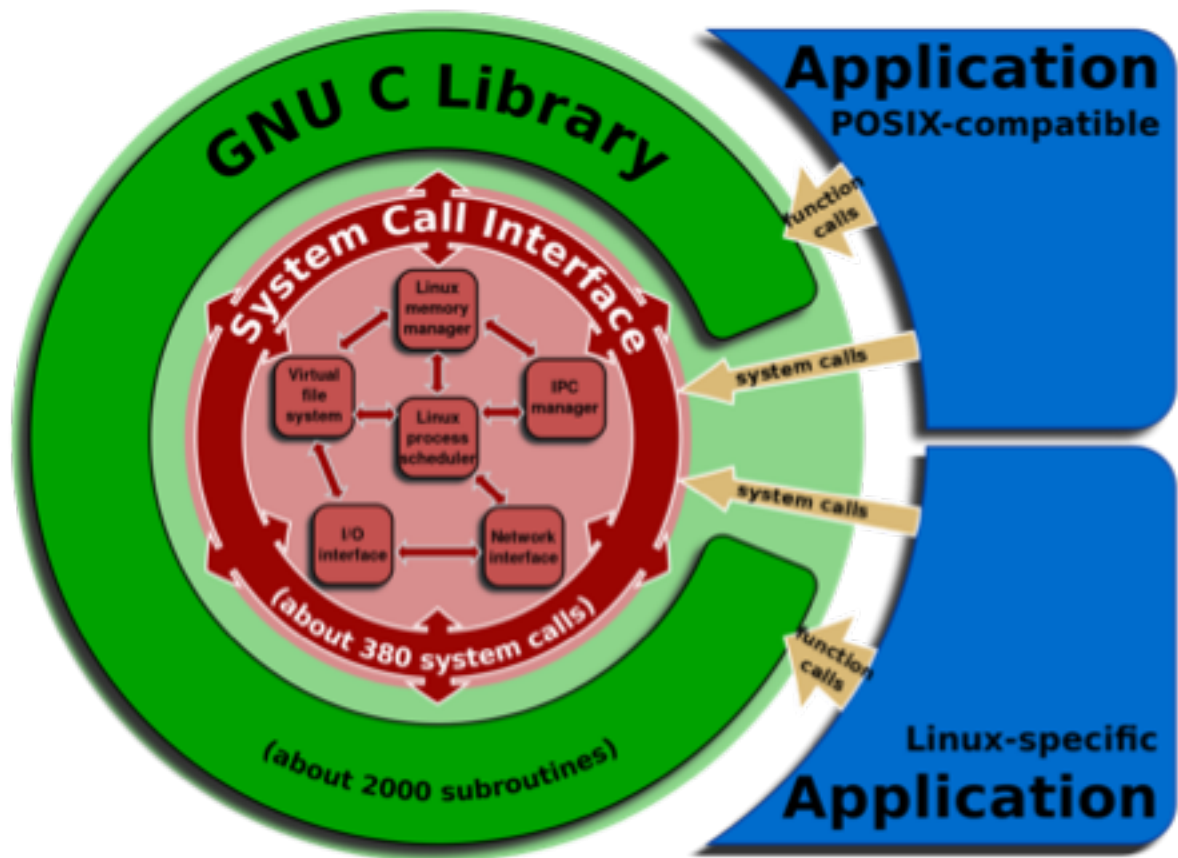
DESCRIPTION top

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)

BONUS

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



System Call



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() InitializeSecurityDescriptor() 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

Programming Language Support

Compilers, assemblers, debuggers and interpreters sometimes provided

Program Loading and Execution

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

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

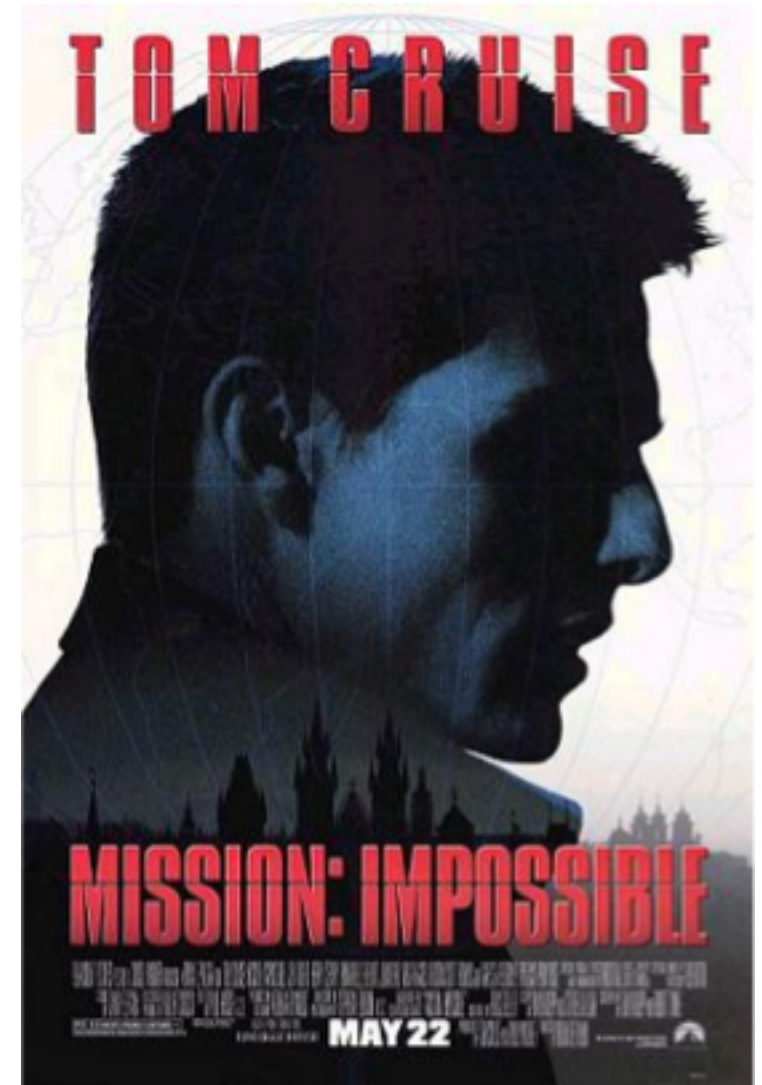
Operating System Design and Implementation

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

Define the User/System Goals

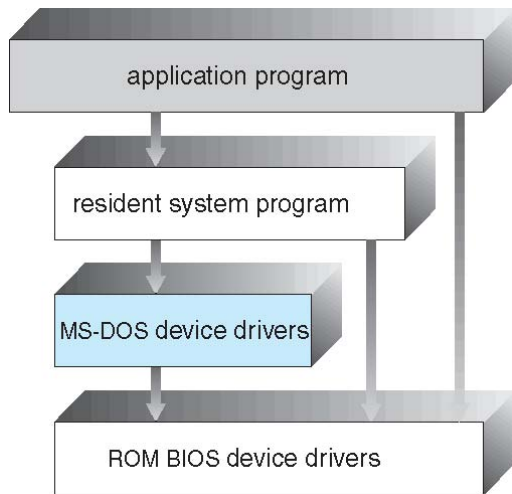
Define the Policies (**What** will be done?) and Mechanisms (**How** to do it?)

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

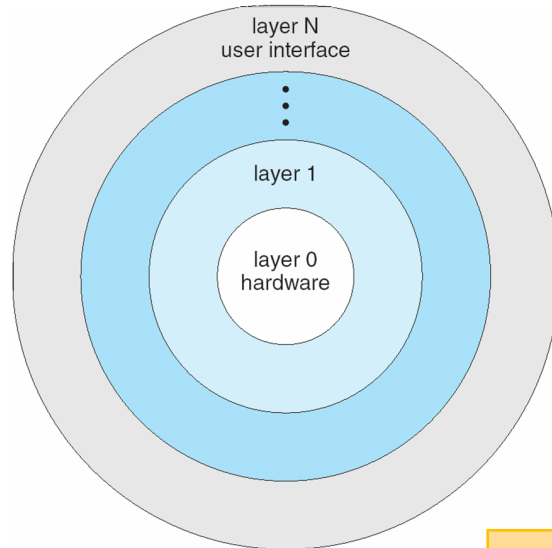


Operating System Structure

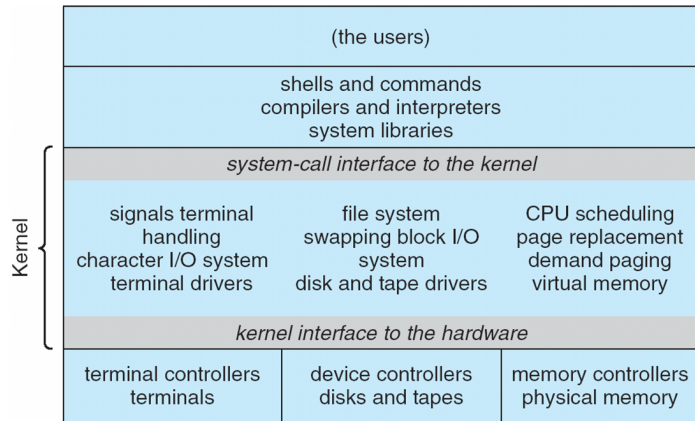
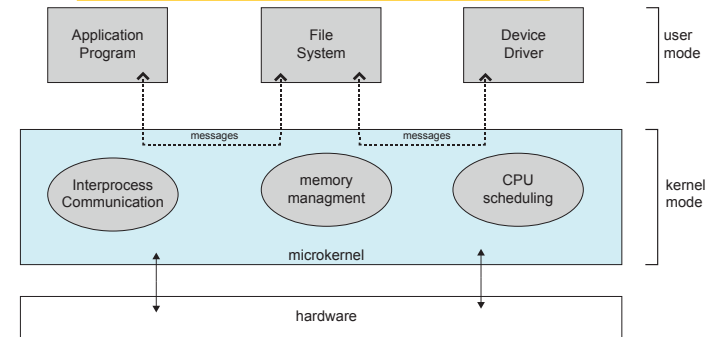
Simple Structure



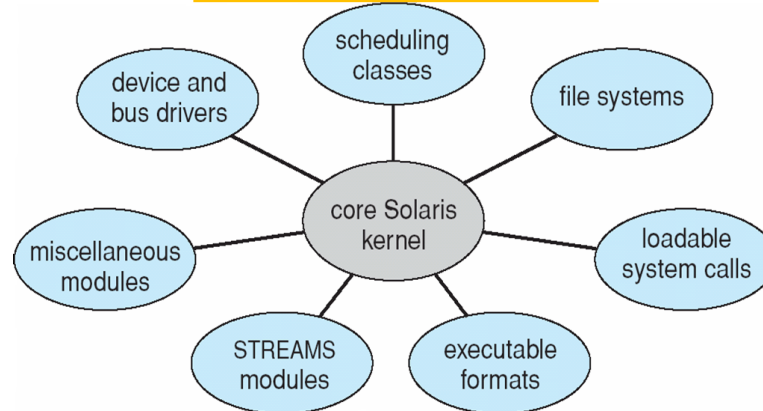
Layered Structure



Microkernel Structure

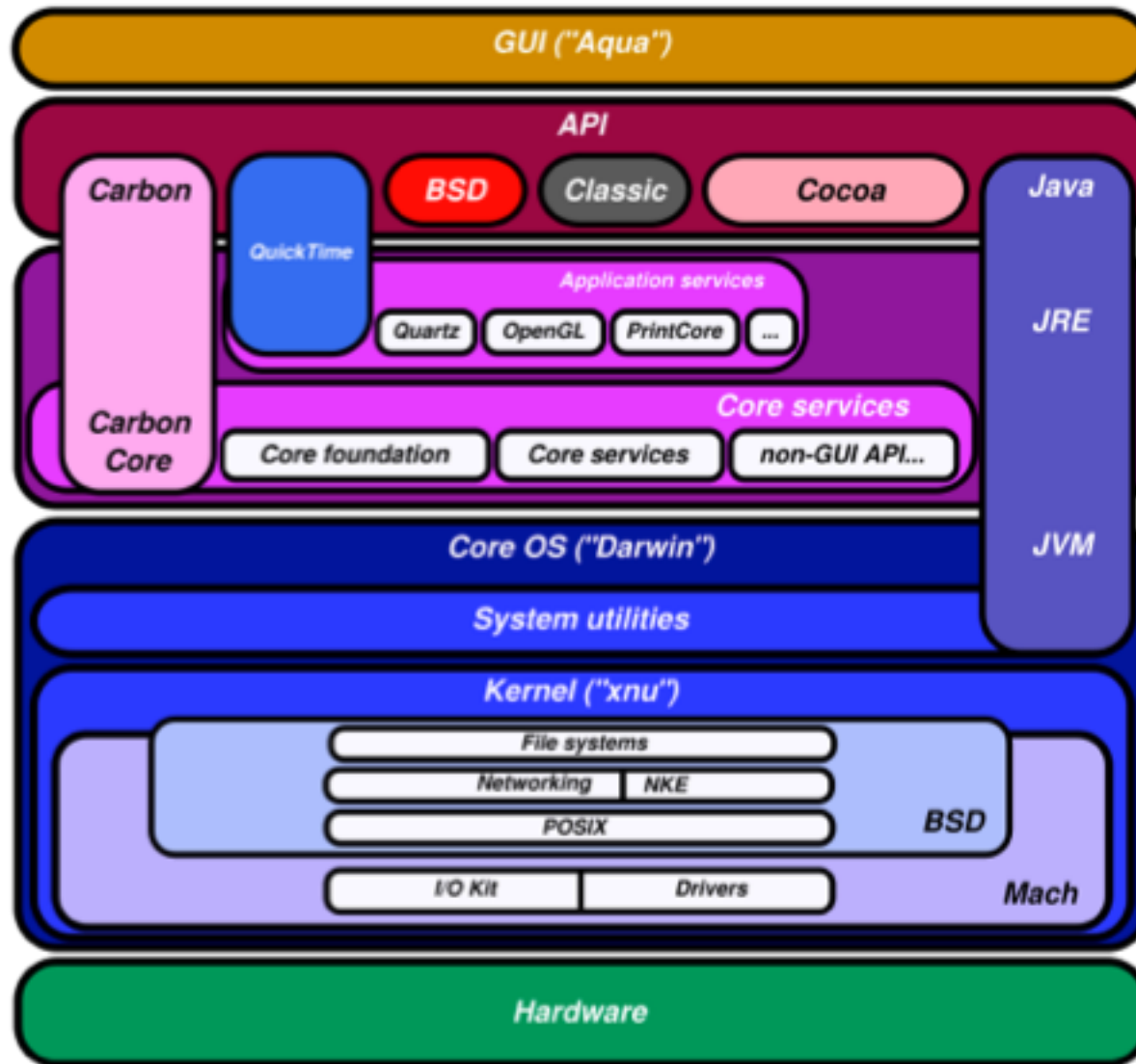


Modular Structure

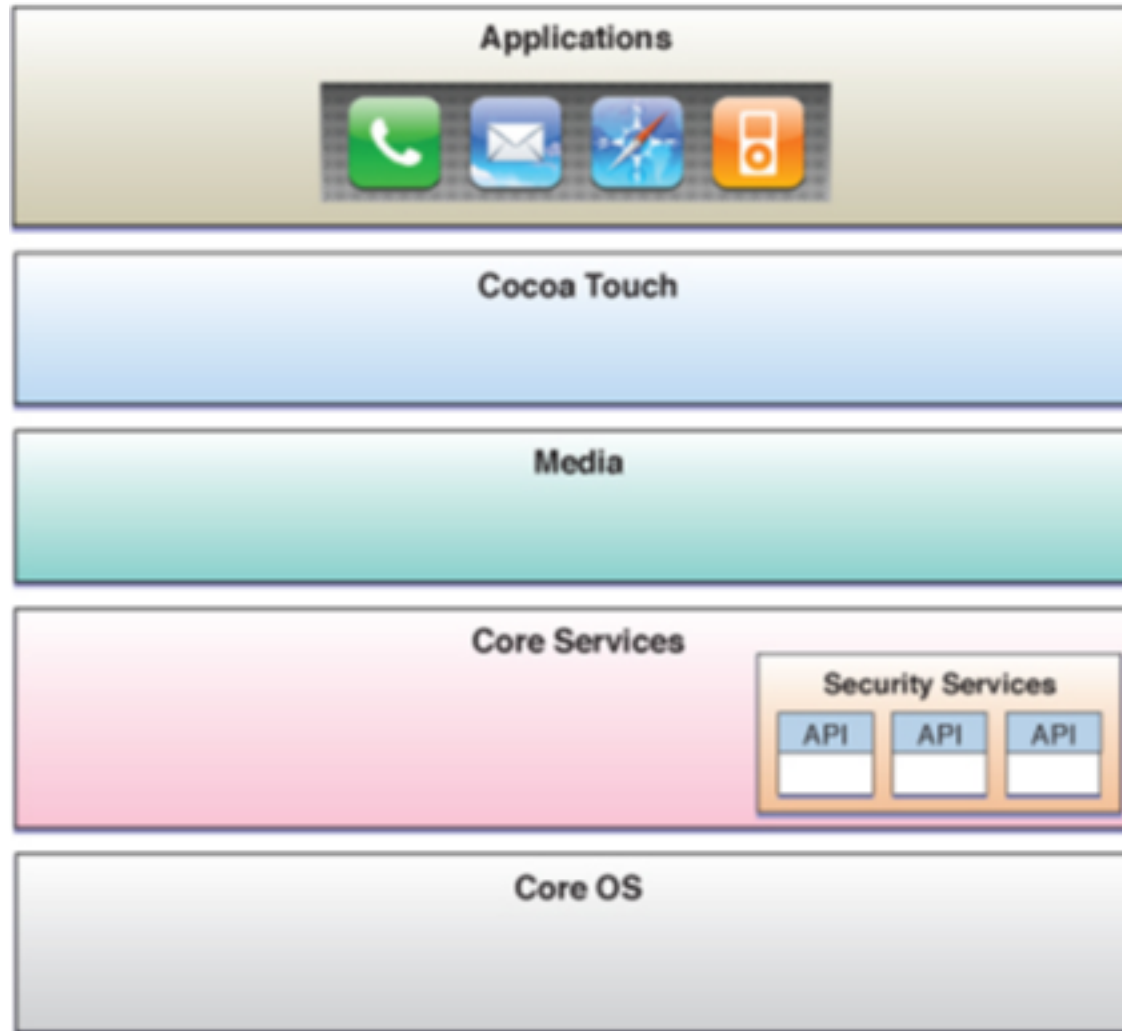


Hybrid Structure

Mac OS Structure



iOS Structure



Android Structure



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

Operating System Debugging

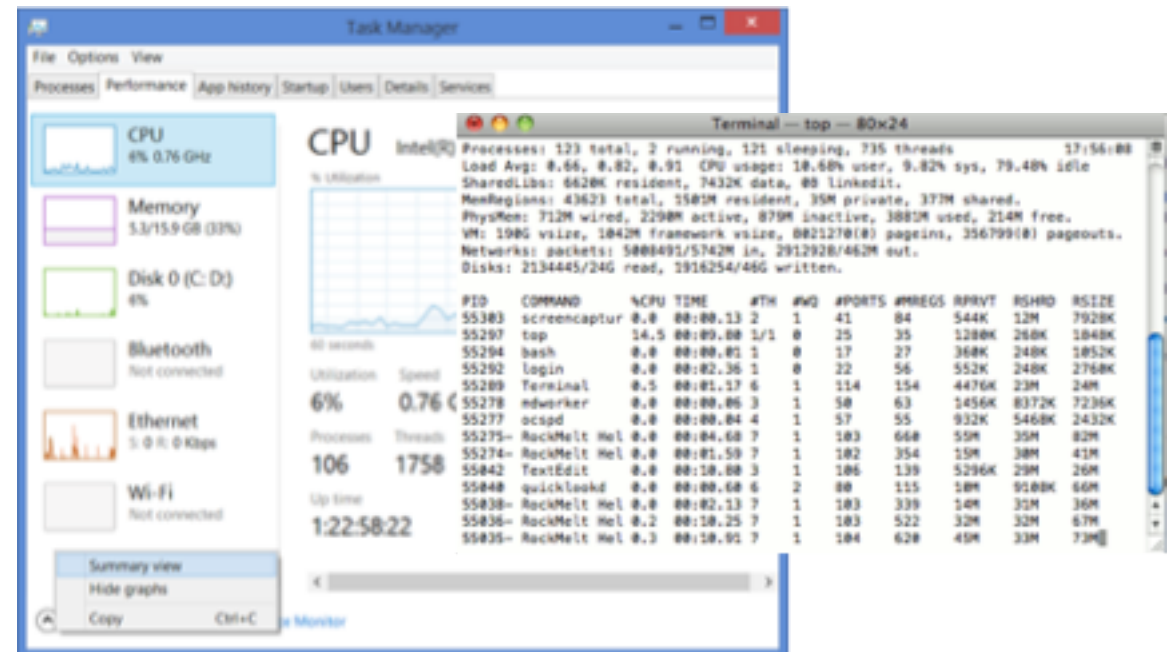
Kernighan's Law: "Debugging is twice as hard as writing the code in the first place. Therefore, if you write the code as cleverly as possible, you are, by definition, not smart enough to debug it."

OS generates *log* files containing error information

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

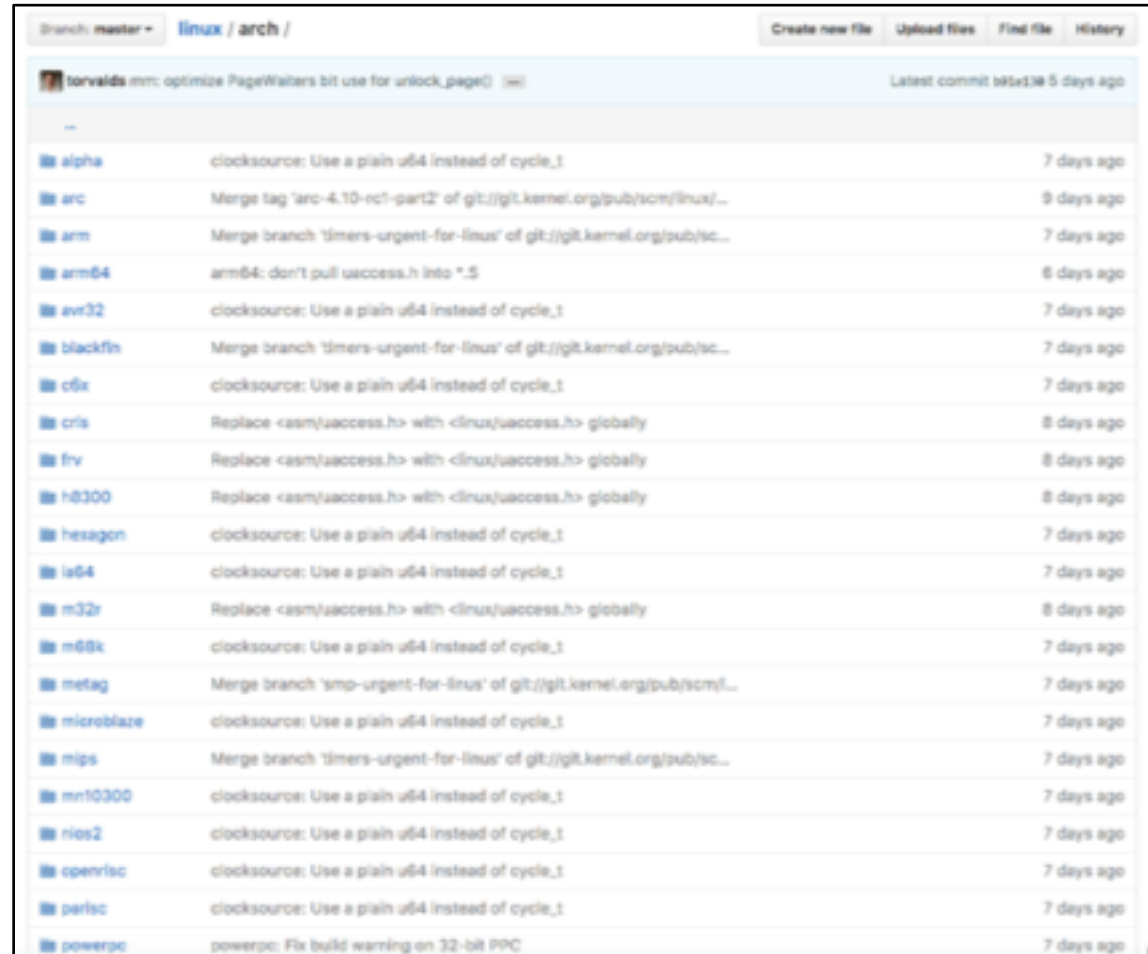
OS failure can generate **crash dump** file containing kernel memory

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



The screenshot shows the GitHub repository for the Linux kernel, specifically the 'arch' directory. The repository is named 'linux / arch' and is on the 'master' branch. The latest commit is by 'torvalds' with the message 'mm: optimize PageWalters bit use for unlock_page()' and was committed 5 days ago. The table below lists the various hardware architectures supported by the kernel, each with a brief description of the commit and the time since the last update.

Architecture	Commit Description	Last Update
alpha	clocksource: Use a plain u64 instead of cycle_1	7 days ago
arc	Merge tag 'arc-4.10-rc1-part2' of git://git.kernel.org/pub/scm/linux/...	9 days ago
arm	Merge branch 'timers-urgent-for-linux' of git://git.kernel.org/pub/sc...	7 days ago
arm64	arm64: don't pull uaccess.h into *.S	6 days ago
avr32	clocksource: Use a plain u64 instead of cycle_1	7 days ago
blackfin	Merge branch 'timers-urgent-for-linux' of git://git.kernel.org/pub/sc...	7 days ago
cfxk	clocksource: Use a plain u64 instead of cycle_1	7 days ago
cris	Replace <asm/uaccess.h> with <linux/uaccess.h> globally	8 days ago
frv	Replace <asm/uaccess.h> with <linux/uaccess.h> globally	8 days ago
h8300	Replace <asm/uaccess.h> with <linux/uaccess.h> globally	8 days ago
hexagon	clocksource: Use a plain u64 instead of cycle_1	7 days ago
ia64	clocksource: Use a plain u64 instead of cycle_1	7 days ago
m32r	Replace <asm/uaccess.h> with <linux/uaccess.h> globally	8 days ago
m68k	clocksource: Use a plain u64 instead of cycle_1	7 days ago
metag	Merge branch 'smo-urgent-for-linux' of git://git.kernel.org/pub/scm/...	7 days ago
microblaze	clocksource: Use a plain u64 instead of cycle_1	7 days ago
mips	Merge branch 'timers-urgent-for-linux' of git://git.kernel.org/pub/sc...	7 days ago
mn10300	clocksource: Use a plain u64 instead of cycle_1	7 days ago
nios2	clocksource: Use a plain u64 instead of cycle_1	7 days ago
openrisc	clocksource: Use a plain u64 instead of cycle_1	7 days ago
parisc	clocksource: Use a plain u64 instead of cycle_1	7 days ago
powerpc	powerpc: Fix build warning on 32-bit PPC	7 days ago

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



キャプテン