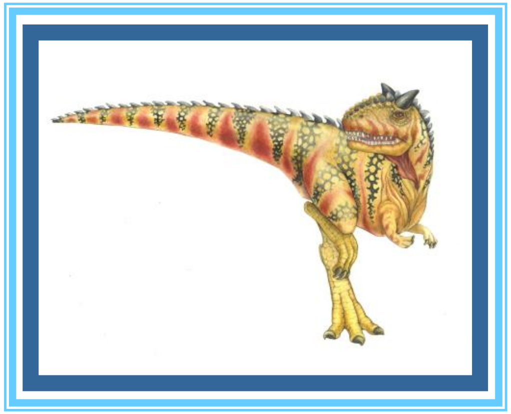
Chapter 2: Operating-System Structures



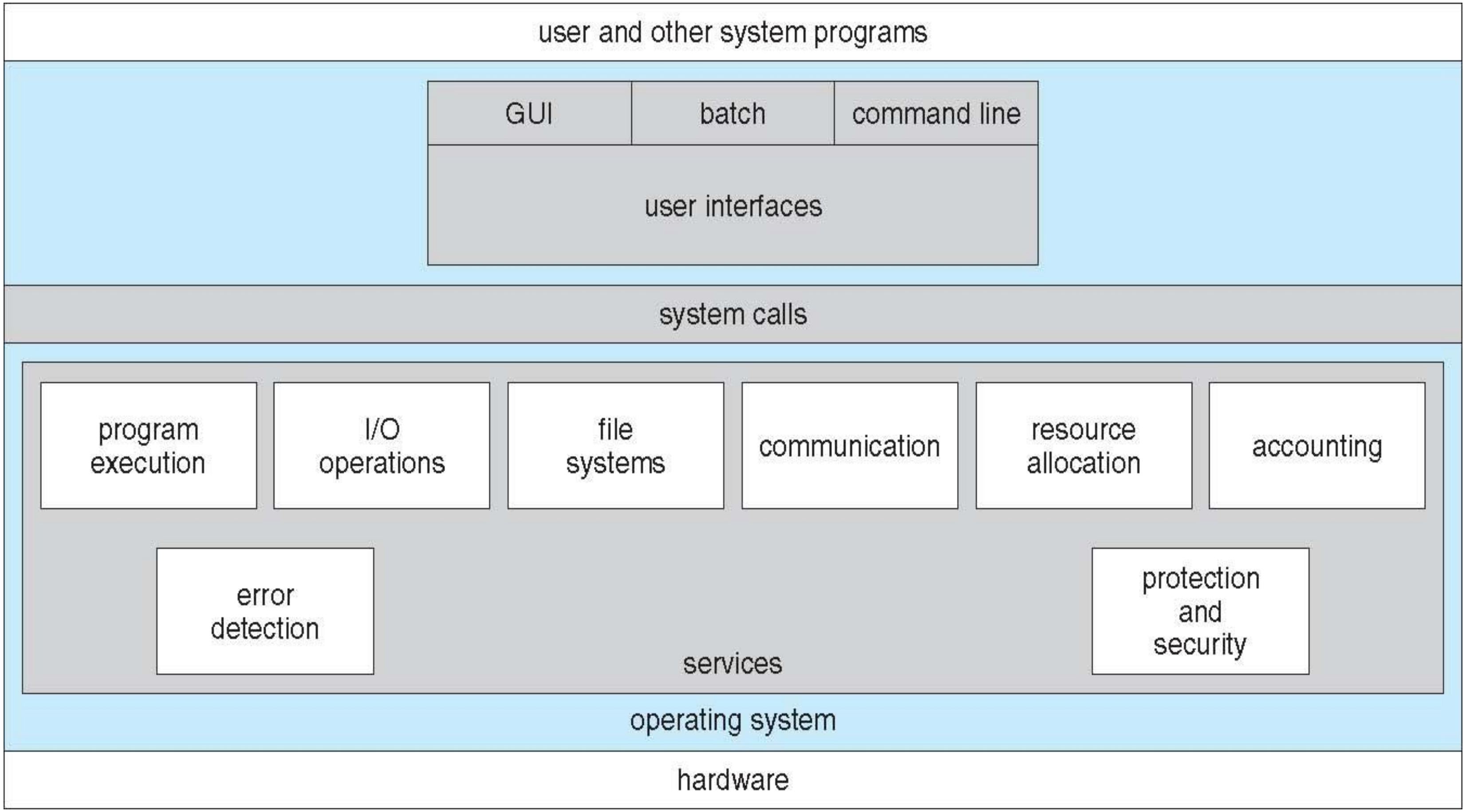
Chapter 2: Outline

- What are the services that the Operating System provides?
  - **From both users & system designers point of view.**
  - **Both may have different requirements.**
- Operating System User Interfaces (UI).
- System Calls.
  - Types of System Calls.
- Operating System Design and Structure.
- Operating System Debugging/Troubleshooting.

Objectives of Chapter 2

- To describe the services an operating system provides to users, processes, and other systems.
- To discuss system calls.
- To discuss the various ways of structuring an operating system.
  - Will discuss several designs.
- Apply tools for monitoring operating system performance.
  - For billing users.
  - For keeping track of usage.
  - For diagnosing problems.

A View of Operating System Services



Operating System Services

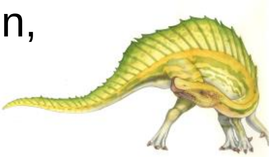
- *From chapter #1: operating systems provide an environment within which programs can be executed.*
- An OS can be viewed from various angles.
- **One angle: from the point of view of the services that it provides.**
- **One set of operating-system services provides functions that are helpful to the user**:
  - **User interface** - Almost all operating systems have a user interface (**UI**).
    - **Command-Line (CLI), Graphics User Interface (GUI), Batch interface (minimal interaction).**
  - **Program execution** - The system must be able to load a program into memory and to run that program, end execution, either normally or abnormally (indicating error).
  - **I/O operations** - A running program may require I/O, which may involve a file or an I/O device.
    - Recall that no direct control of I/O devices given to user.

Operating System Services (Cont.)

- **OS services useful to the user (continued):**
  - **File-system** - Programs need to read and write files and directories, create and delete them, search them, list file Information, permission management. Hence, the OS needs to support a file-system.
  - **Communications** – Processes may exchange information, on the same computer or between computers over a network.
    - Communications may be via shared memory or through message passing (packets moved by the OS).
  - **Error detection** – OS needs to be constantly aware of possible errors.
    - May occur in the CPU (example?), in memory (example?), in I/O devices (example?), in user program (example?).
    - For each type of error, OS should take the appropriate action.
    - May sometimes have to halt the process, or the system.
    - Example errors:
      - » File path not found.
      - » Access denied.
      - » Disk full.
      - » Stack overflow.

Operating System Services (Cont.)

- **Another set of OS functions exists for ensuring the efficient operation of the system itself via resource sharing:**
  - **Resource allocation -** When multiple users or multiple jobs running concurrently, resources must be allocated to each of them
    - Many types of resources -   CPU time, main memory, file storage, I/O devices.
    - Scheduling algorithms are in place for allocating CPU time.
  - **Accounting -** To keep track of which users use how much and what kinds of computer resources: **task manager in Windows, Activity monitor in Apple, top utility in Linux.**
    - May be used for billing purposes, to check for quota, or to improve some aspect of the OS by reconfiguring the system.
  - **Protection and security -** The owners of information stored in a multiuser or networked computer system may want to control use of that information, concurrent processes should not interfere with each other
    - **Protection** involves ensuring that all access to system resources is controlled.
    - **Security** of the system from outsiders requires user authentication, extends to defending external I/O devices from invalid access attempts.

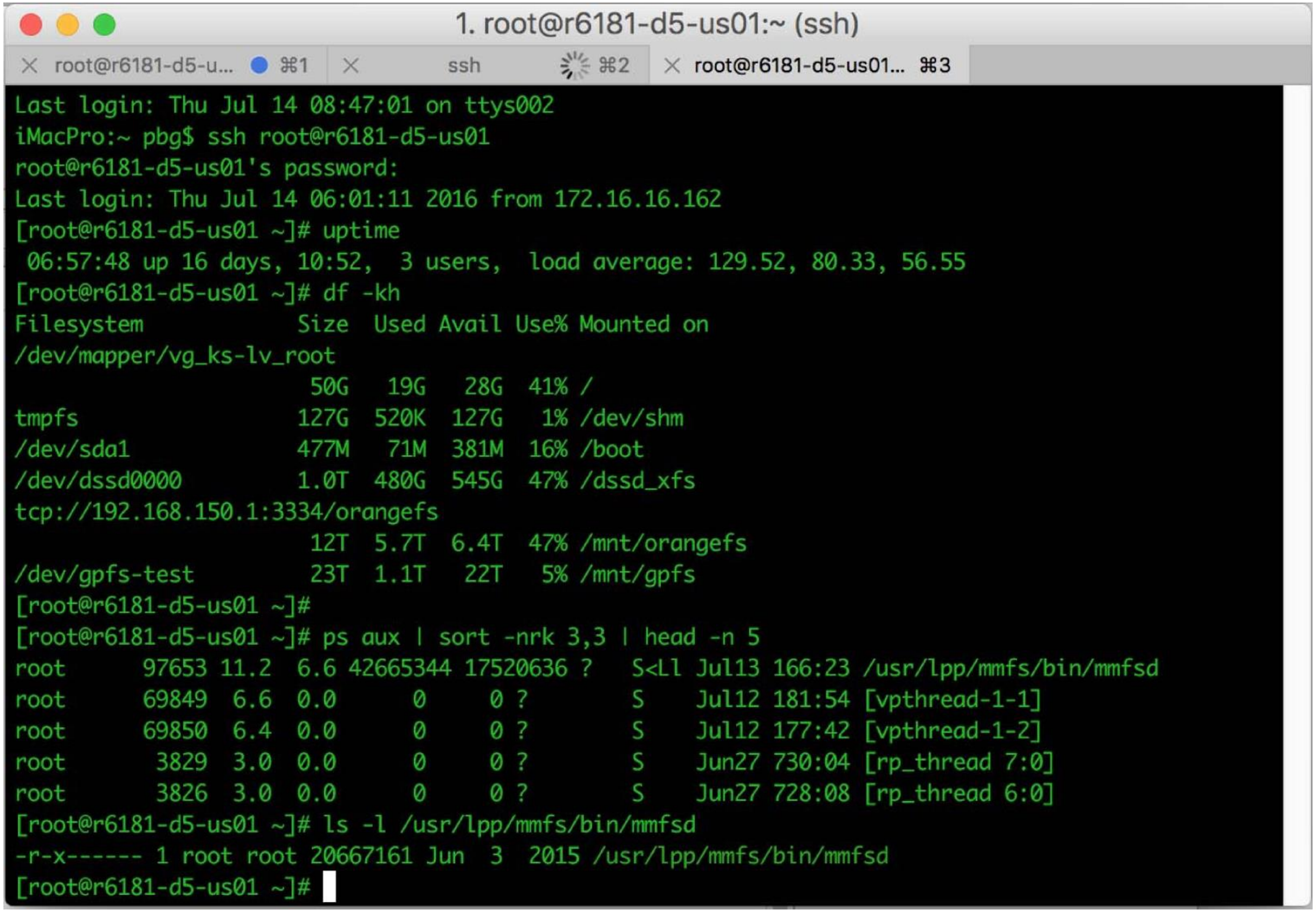


User Interfaces (UI) to the OS

- **UI-1: Command line interface, aka a command interpreter.**
- Provided by Microsoft (DOS prompt), Linux & OSX (terminal).
- Also called a **shell**, many to choose from: Bourne shell, Korn shell etc.
- **Main function of a shell?**
- **To get & execute the next user specified command.**
- Commands are basically programs that perform many tasks: create, delete, list, print, copy, move, etc.
- We have a system prompt, that waits for the next user command.
- For example, we have: **rm file.txt**.
- “rm” here is the name of a program that is used to delete files.
- On seeing this input, shell locates the code of the program named “rm”, loads it to memory & executes it with one argument, the name of the file.
- It then waits for the next user command.

- As such, the shell does not understand the command in any way.
- It merely uses the command to identify a file to be loaded into memory & executed.
- The programs corresponding to these files are located in the /bin directory.
- **To see the list of executable programs, simply type the following in the command line: ls /bin.**

Bourne Shell

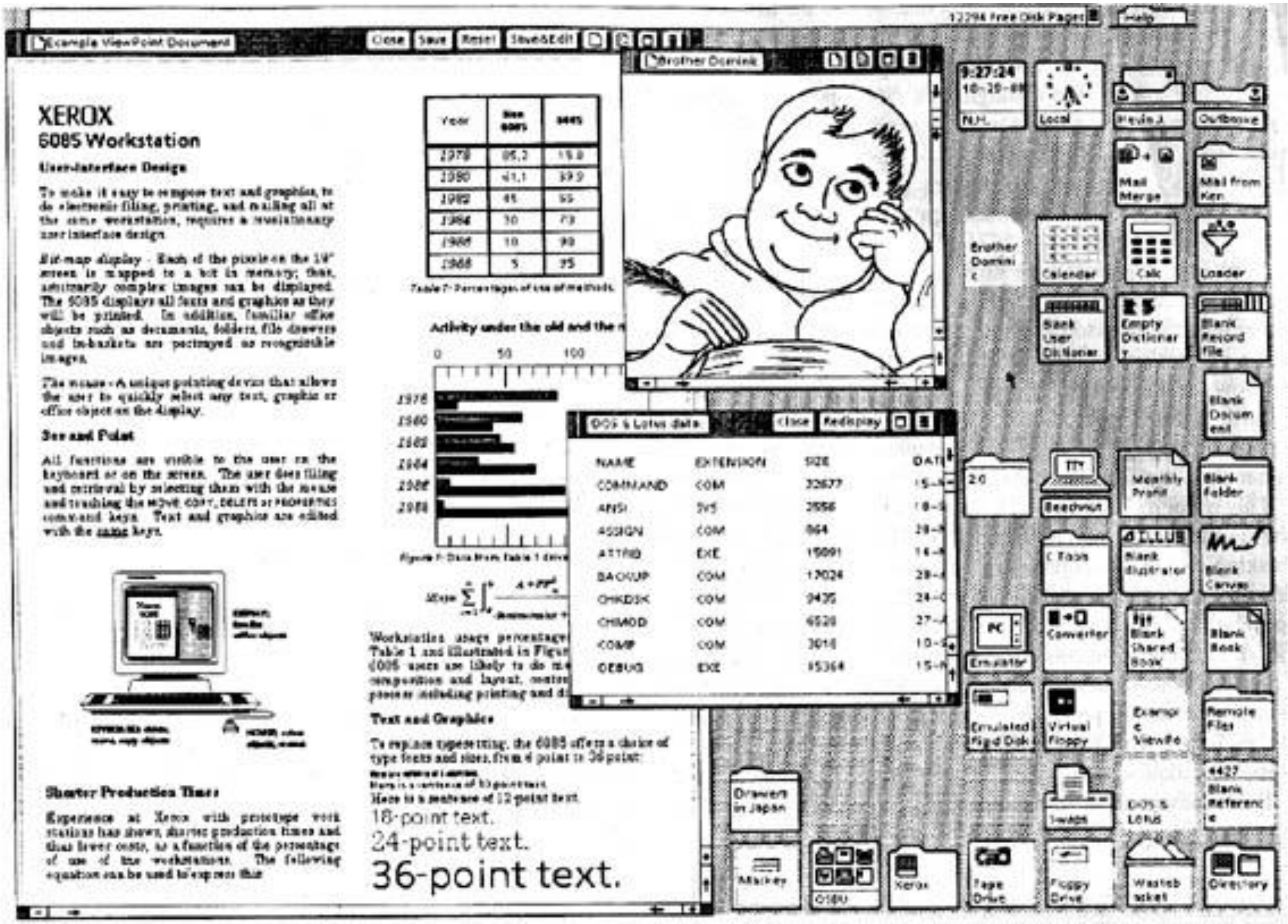


```
Last login: Thu Jul 14 08:47:01 on ttys002
iMacPro:~ pbg$ ssh root@r6181-d5-us01
root@r6181-d5-us01's password:
Last login: Thu Jul 14 06:01:11 2016 from 172.16.16.162
[root@r6181-d5-us01 ~]# uptime
 06:57:48 up 16 days, 10:52,  3 users,  load average: 129.52, 80.33, 56.55
[root@r6181-d5-us01 ~]# df -kh
Filesystem            Size  Used Avail Use% Mounted on
/dev/mapper/vg_ks-lv_root
                       50G   19G   28G  41% /
tmpfs                 127G  520K  127G   1% /dev/shm
/dev/sda1             477M   71M  381M  16% /boot
/dev/dssd0000         1.0T  480G  545G  47% /dssd_xfs
tcp://192.168.150.1:3334/orangefs
                       12T  5.7T  6.4T  47% /mnt/orangefs
/dev/gpfs-test         23T  1.1T   22T   5% /mnt/gpfs
[root@r6181-d5-us01 ~]#
[root@r6181-d5-us01 ~]# ps aux | sort -nrk 3,3 | head -n 5
root     97653 11.2  6.6 42665344 17520636 ?   S<Ll Jul13 166:23 /usr/lpp/mmfs/bin/mmfsd
root     69849  6.6  0.0      0     0 ?        S    Jul12 181:54 [vpthread-1-1]
root     69850  6.4  0.0      0     0 ?        S    Jul12 177:42 [vpthread-1-2]
root      3829  3.0  0.0      0     0 ?        S    Jun27 730:04 [rp_thread 7:0]
root      3826  3.0  0.0      0     0 ?        S    Jun27 728:08 [rp_thread 6:0]
[root@r6181-d5-us01 ~]# ls -l /usr/lpp/mmfs/bin/mmfsd
-r-x------ 1 root root 20667161 Jun  3  2015 /usr/lpp/mmfs/bin/mmfsd
[root@r6181-d5-us01 ~]# 
```

User Operating System Interface - GUI

- UI-2: User-friendly desktop metaphor interface
  - Usually using mouse, keyboard, and monitor.
  - **Icons** represent files, programs, actions, etc.
  - Various mouse actions over objects in the interface cause various actions (provide information, options, execute function, open directory (known as a **folder**).
  - Who invented the GUI?
  - Invented at Xerox PARC, early 1970s.
  - Popularized (stolen?) by Apple, Microsoft in 1980s.
- Many systems now include both CLI and GUI interfaces.
  - Microsoft Windows is GUI with CLI “command” shell.
  - Apple Mac OS X is “Aqua” GUI interface with UNIX kernel underneath and shells available.
  - Unix and Linux have CLI with optional GUI interfaces (CDE, KDE, GNOME).

Source: http://www.catb.org/esr/writings/taouu/html/ch02s05.html



XEROX
6085 Workstation

User-Interface Design

To make it easy to compose text and graphics, to do electronic filing, printing, and mailing all at the same workstation, requires a revolutionary user interface design.

Bit-map display - Each of the pixels on the 19" screen is mapped to a bit in memory; thus, arbitrarily complex images can be displayed. The 6085 displays all fonts and graphics as they will be printed. In addition, familiar office objects such as documents, folders, file drawers and in-baskets are portrayed as recognizable images.

The mouse - A unique pointing device that allows the user to quickly select any text, graphic or office object on the display.

See and Point

All functions are visible to the user on the keyboard or on the screen. The user does filing and retrieval by selecting them with the mouse and touching the MOVE, COPY, DELETE or PROPERTIES command keys. Text and graphics are edited with the same keys.

Shorter Production Times

Experience at Xerox with prototype work stations has shown shorter production times and thus lower costs, as a function of the percentage of use of the workstations. The following equation can be used to express this

Year	[illegible]	6085
1978	85.2	15.0
1980	61.1	39.9
1982	45	55
1984	30	70
1986	10	90
1988	5	95

Table 1: Percentages of use of methods.

Activity under the old and the n

Figure 1: Data from Table 1 deriv

Workstation usage percentage
Table 1 and illustrated in Figur
6085 users are likely to do m
composition and layout, conte
process including printing and d

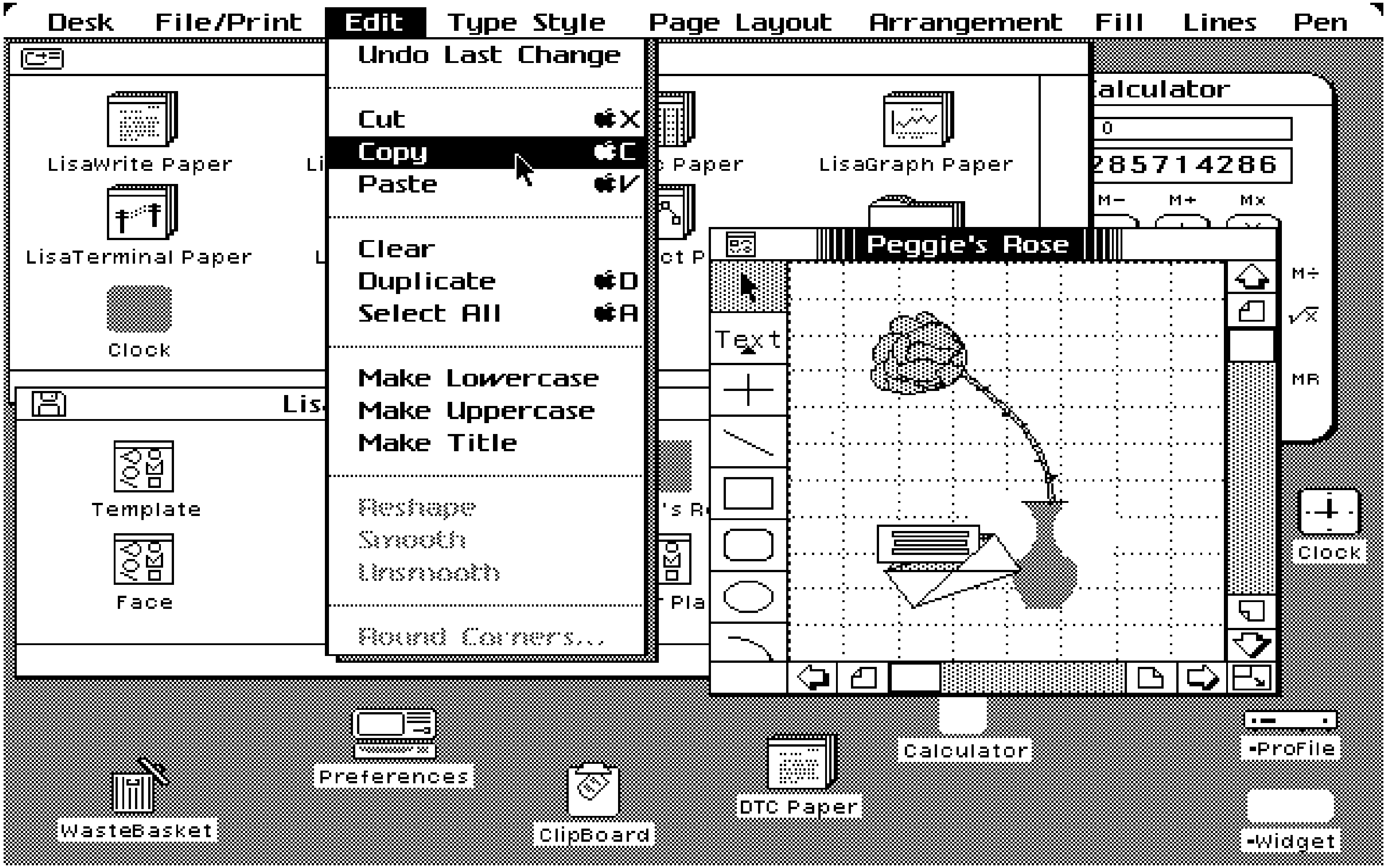
Text and Graphics

To replace typesetting, the 6085 offers a choice of type fonts and sizes, from 4 point to 36 point:

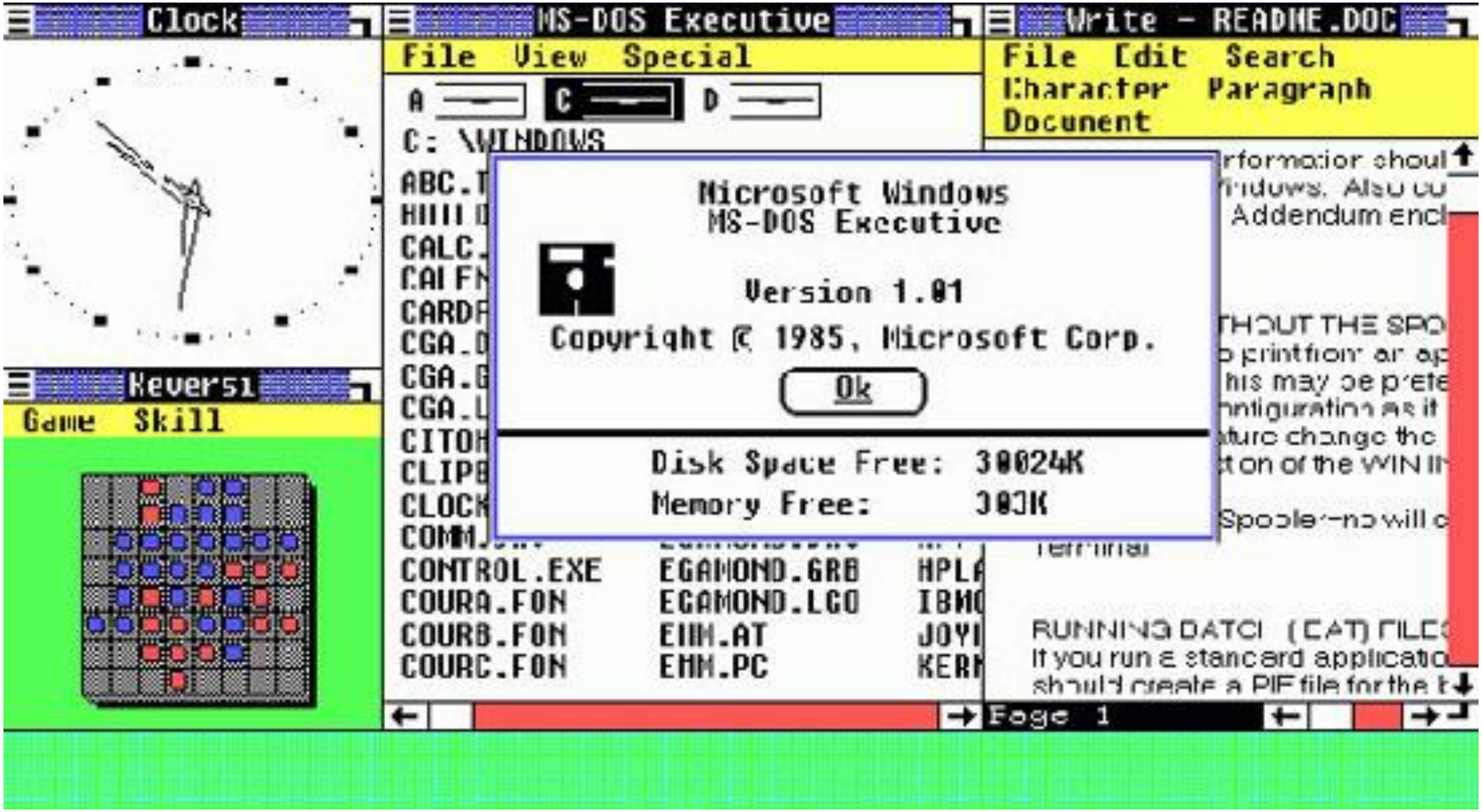
[illegible]
Here is a sentence of 10-point text.
Here is a sentence of 12-point text.
18-point text.
24-point text.
36-point text.

NAME	EXTENSION	SIZE	DAT
COMMAND	COM	32677	15-[illegible]
ANSI	SYS	2556	18-[illegible]
ASSIGN	COM	864	28-[illegible]
ATTRIB	EXE	15091	14-[illegible]
BACKUP	COM	17024	28-[illegible]
CHKDSK	COM	9435	24-[illegible]
CHMOD	COM	6528	27-[illegible]
COMP	COM	3016	10-[illegible]
DEBUG	EXE	15364	15-[illegible]

https://i1.wp.com/www.mac-history.de/wp-content/uploads/2008/09/apple_lisa_screenshot.gif



http://history-computer.com/ModernComputer/Software/images/Windows1.jpg

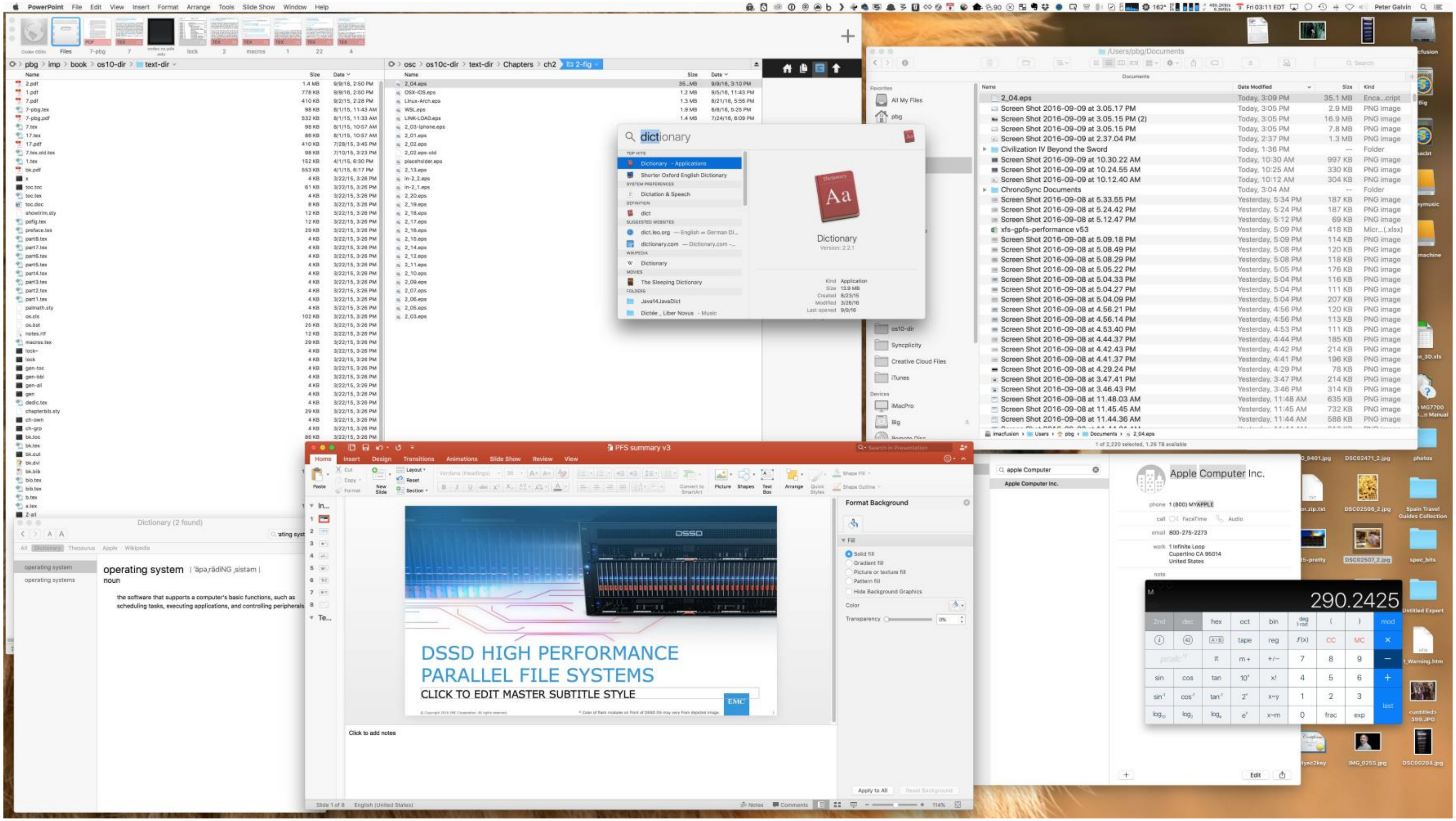

Microsoft Windows
MS-DOS Executive

Version 1.01

Copyright © 1985, Microsoft Corp.

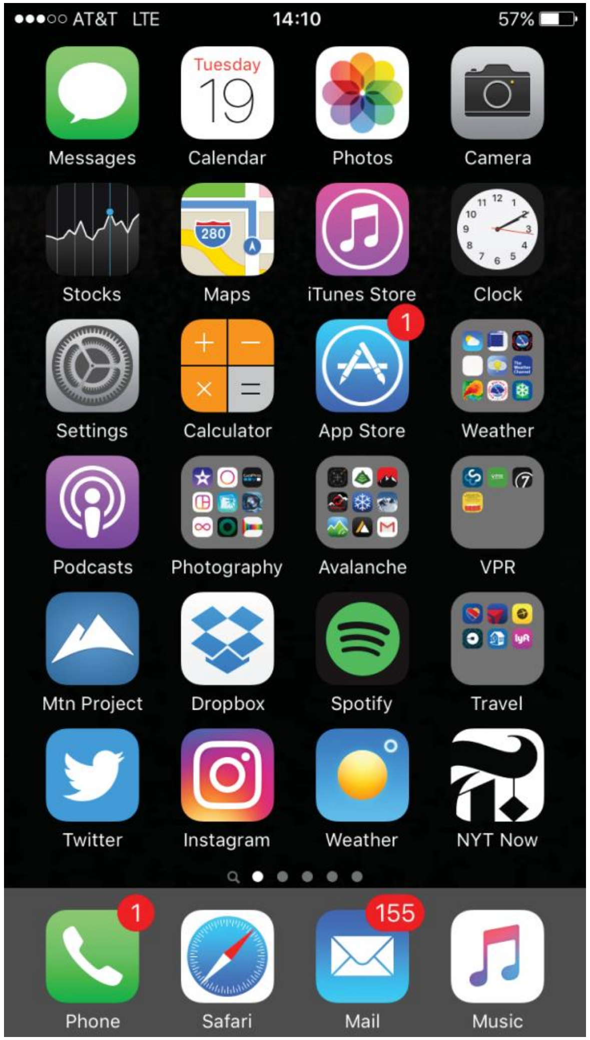
Disk Space Free: 30024K
Memory Free: 303K

The Mac OS X GUI


PowerPoint File Edit View Insert Format Arrange Tools Slide Show Window Help
Peter Galvin
pbg > imp > book > os10-dir > text-dir
osc > os10c-dir > text-dir > Chapters > ch2 > 2-fig
dictionary
Dictionary
/Users/pbg/Documents
Dictionary (2 found)
operating system
PFS summary v3
DSSD HIGH PERFORMANCE PARALLEL FILE SYSTEMS
CLICK TO EDIT MASTER SUBTITLE STYLE
Click to add notes
Format Background
Apple Computer Inc.
290.2425

Touchscreen Interfaces

- Touchscreen devices require new interfaces
  - Mouse not possible or not desired
  - Actions and selection based on gestures: tap, swipe.
  - Virtual keyboard for text entry.
  - Early devices had physical keyboards.
- Voice commands.
  - Siri, Cortana, Google.



Choice of Interface – CL or GUI?

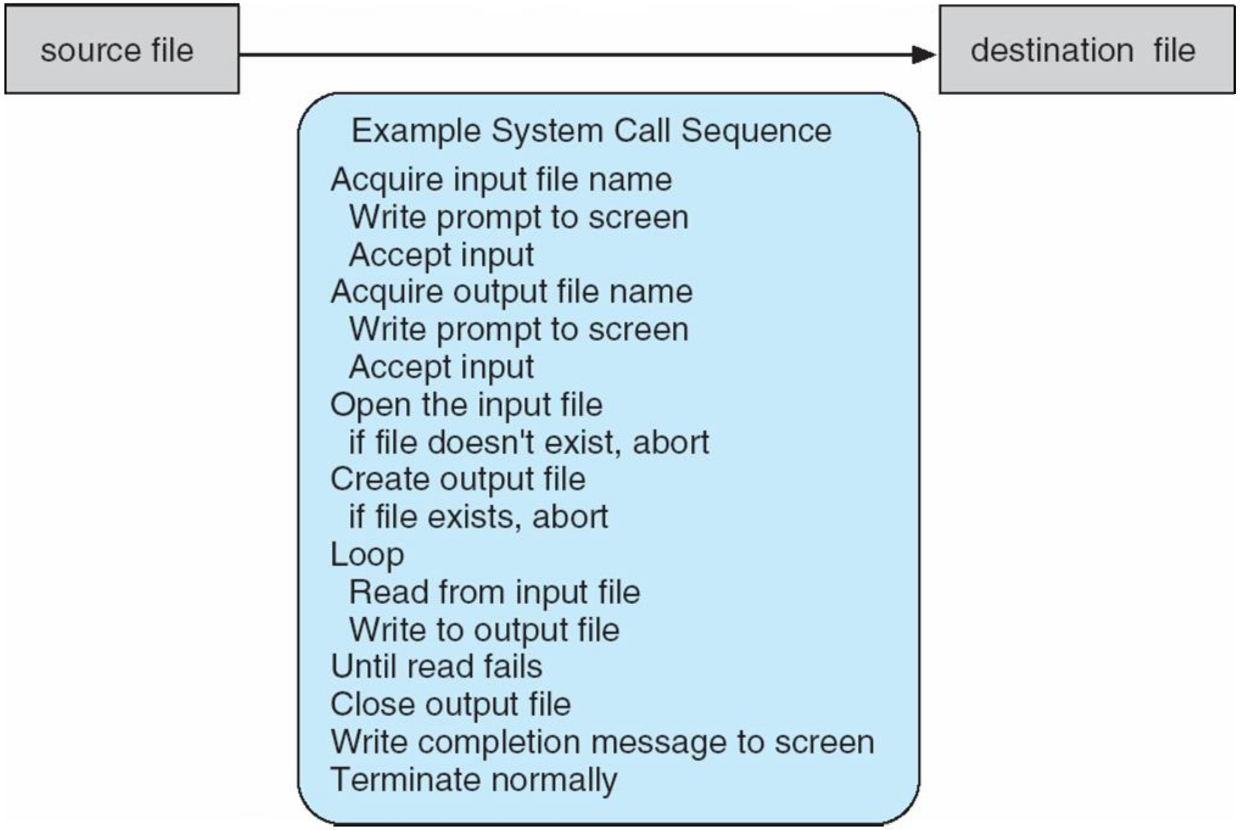
- Personal preference.
- Both options are provided by Microsoft Windows, Linux & OSX.
- **Advantage of GUI?**
- User friendly.
- **Advantage of CL?**
- More powerful, access to more features.
- Generally preferred by users with a good knowledge of computer operations: system administrators, programmers, CS students.
- The UI is typically not a part of the OS structure, so we will not study it in this course.

System Calls

- **What are system calls?**
- **They provide an interface to the services (discussed earlier) made available by the OS.**
- Program needs an OS service => need to make 1, or more system calls.
- **These are available (to programmers/app developers) as functions, written in a high level language, such as C.**
- Programmers can write programs that use system calls.
- OS intercepts these calls & executes relevant instructions.
- Let us motivate the discussion by looking at an example.
- **Common task – a program that reads data from one file & copies it to another file.**
- This task can be split up into many sub-tasks, as shown in the next slide.
- Each task → a system call.
- Note: may have several error conditions in this task (need to deal with all of them).

Example of System Calls

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



Example System Call Sequence

```
Acquire input file name
  Write prompt to screen
  Accept input
Acquire output file name
  Write prompt to screen
  Accept input
Open the input file
  if file doesn't exist, abort
Create output file
  if file exists, abort
Loop
  Read from input file
  Write to output file
Until read fails
Close output file
Write completion message to screen
Terminate normally
```

Some Examples of Linux System Calls

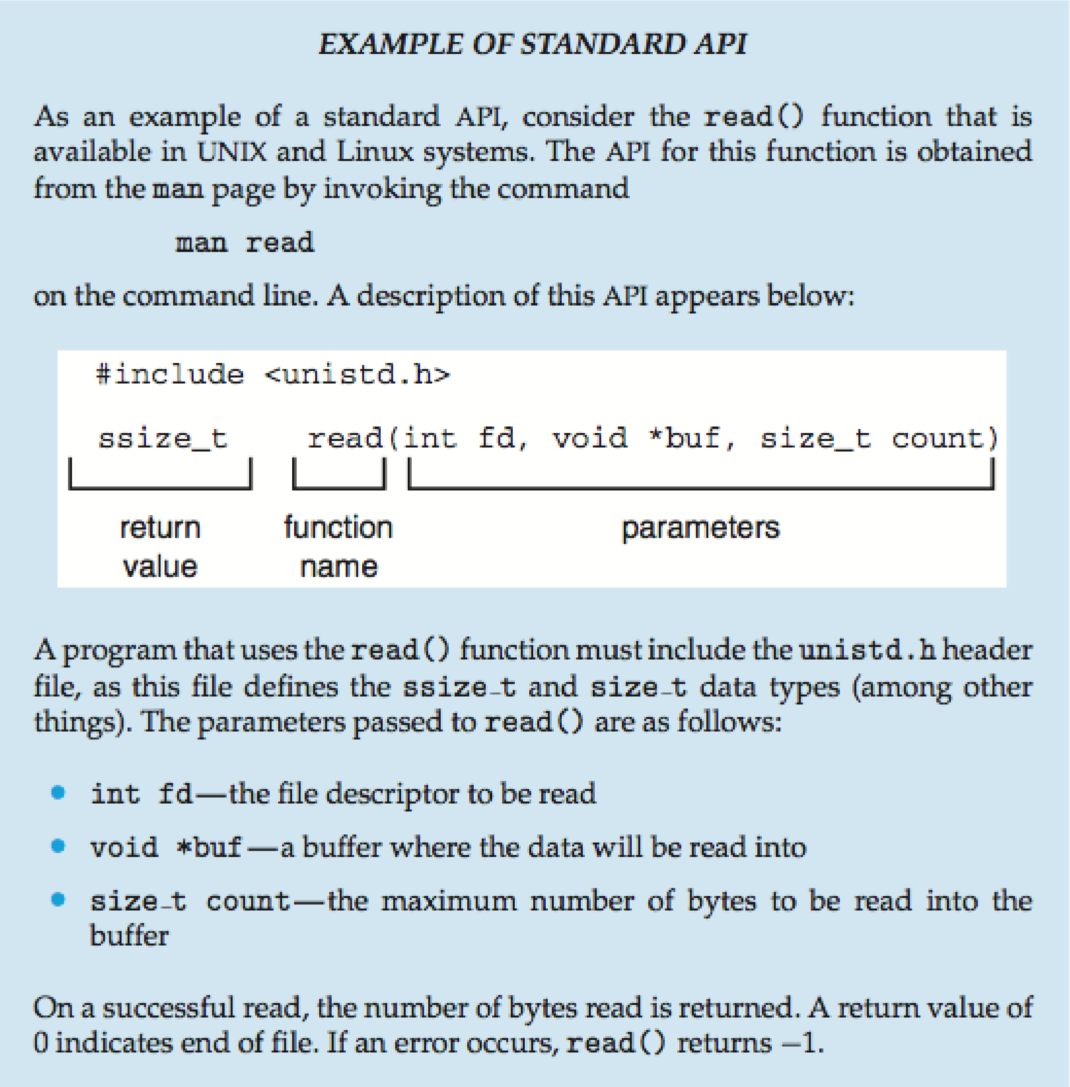
- http://asm.sourceforge.net/syscall.html.
- man7.org/linux/man-pages/man2/syscalls.2.html
- fork ( ): create a new process.
- exit ( ): terminate process.
- read ( ): file read.
- write ( ): file write.
- open ( ): open file.
- close ( ): close file.
- time ( ).
- chmod ( ): change file mode bits (permissions)
- Many more, visit top link for the list.

System Calls

- For the benefit of application programmers (us), accessed by programs via a high-level **Application Programming Interface (API)** rather than direct system call use.
- **Programmers call an API (set of functions), which invoke the system calls on their behalf.**
- **Actual system calls are more complex, so it is easier for programmers to work with the API instead.**
- Common APIs: are Win32 API for Windows, POSIX API for POSIX-based systems (including virtually all versions of UNIX, Linux, and Mac OS X).
- **POSIX: Portable Operating System Interface for compatibility between UNIX based systems.**
- OS provides a library of these API functions to the programmers.
- For Linux, called – libc: https://www.gnu.org/software/libc/manual/

Note that the system-call names used throughout this text are generic, different operating systems may have different names for the same system calls.

Example of Standard API



EXAMPLE OF STANDARD API

As an example of a standard API, consider the `read()` function that is available in UNIX and Linux systems. The API for this function is obtained from the `man` page by invoking the command

```
man read
```

on the command line. A description of this API appears below:

```
#include <unistd.h>

ssize_t    read(int fd, void *buf, size_t count)
```

return value

function name

parameters

A program that uses the `read()` function must include the `unistd.h` header file, as this file defines the `ssize_t` and `size_t` data types (among other things). The parameters passed to `read()` are as follows:

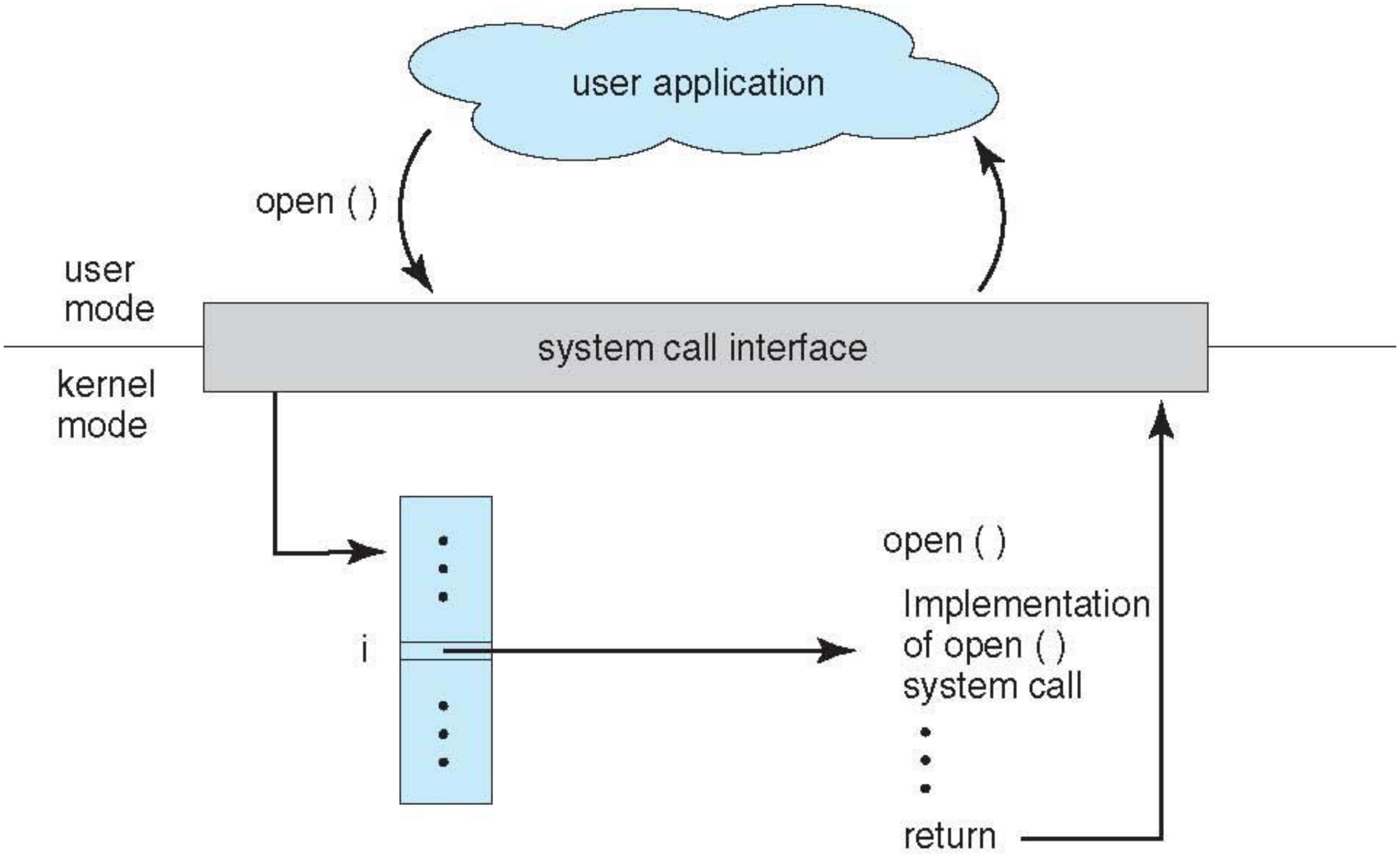
- `int fd`—the file descriptor to be read
- `void *buf`—a buffer where the data will be read into
- `size_t count`—the maximum number of bytes to be read into the buffer

On a successful read, the number of bytes read is returned. A return value of 0 indicates end of file. If an error occurs, `read()` returns −1.

System Call Implementation

- Typically, a number associated with each system call
    - A table is maintained, that is indexed according to these numbers.
- The intended system call is invoked in OS kernel and returns status of the system call and any return values
- **The caller (user/programmer) 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.
    - Details of OS implementation hidden from programmer by API.

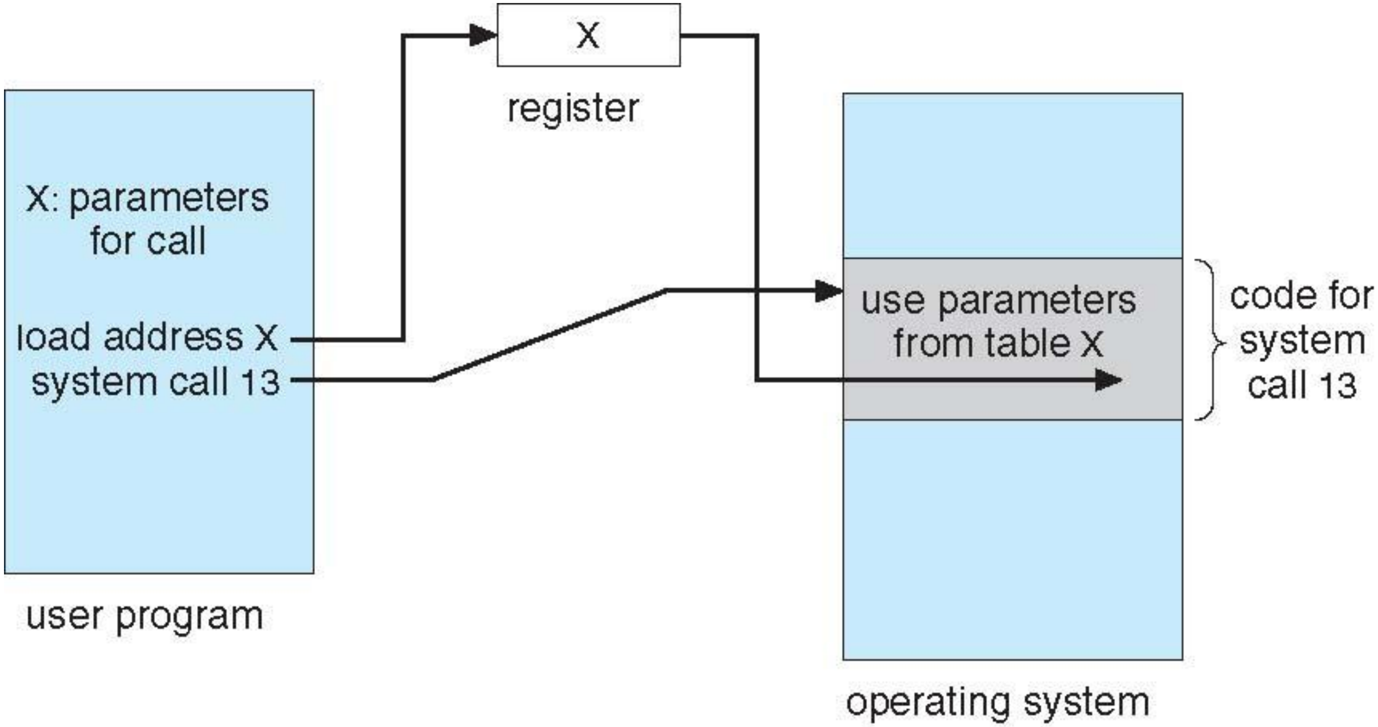
API – System Call – OS Relationship



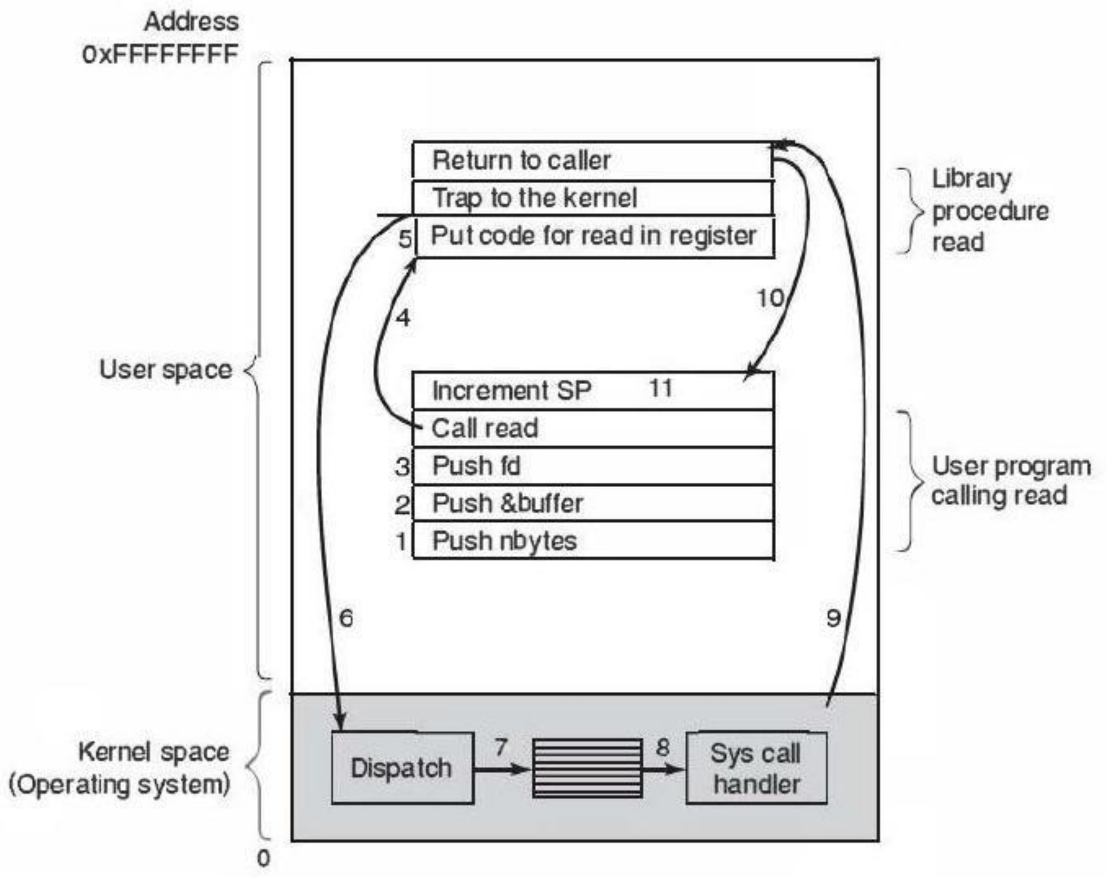
System Call Parameter Passing

- Often, more information is required than simply identity of desired system call, for example, some parameters.
- For example, read a file named chapter2.ppt. What’s the parameter?
  - Exact type and amount of information vary according to OS and call.
- Three general methods used to pass parameters to the OS.
  - Simplest: pass the parameters in registers. Any problem?
    - In some cases, may be more parameters than can fit in registers.
  - Parameters stored in a block, or table, in memory, and address of block passed as a parameter in a register.
    - This approach taken by Linux and Solaris.
  - Parameters may be placed, or **pushed**, onto the **stack** by the program and **popped** off the stack by the operating system
  - Block and stack methods do not limit the number or length of parameters being passed.

Parameter Passing via Table



Steps in making 'read' system call



The 11 steps in making the system call read(fd, buffer, nbytes).

Steps in execution of ‘read’ system call

- API: count = read(fd, buffer, nbytes);
- In preparation for calling ‘read’ procedure (API), which calls actual read system call, calling program first pushes the parameters onto the stack (steps 1-3).
- Next step is call to the library procedure (step 4).
- System call number is put in a register, where the OS can read from (step 5).
- Next is a trap to the OS (interrupts that requests OS attention, changing mode from user – kernel, step 6).
- Kernel following trap examines system call number (maintained as a table), and dispatches to correct system call handler code (step 7).
- The system call is executed (step 8).
- Control is now returned to user space program & next line of code is executed (steps 9 & 10).
- SP is incremented to overwrite over old contents (no longer needed).

Types of System Calls

- **Process control.**
  - Create process, terminate process.
  - Load & execute process.
  - Get process attributes, set process attributes.
  - Wait, signal.
  - Allocate and free memory.

Types of System Calls

- **File management.**
  - create file, delete file.
  - open, close file.
  - read, write, reposition (rewind or skip to end of file).
  - get and set file attributes.
- **Device management.**
  - request device, release device.
  - read, write.
  - get device attributes, set device attributes.
  - logically attach or detach devices (for external drives).

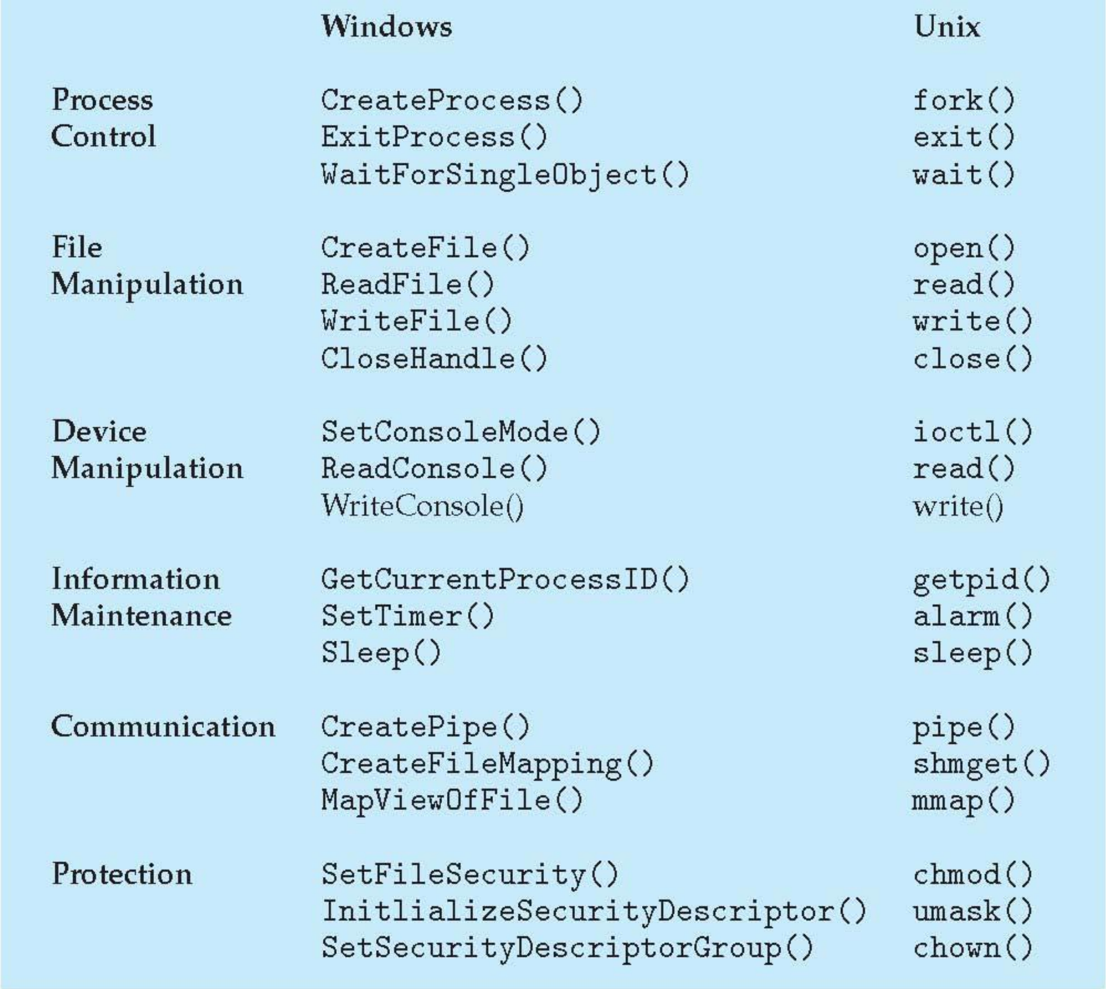
Types of System Calls (Cont.)

- **Information maintenance.**
  - get time or date, set time or date.
  - get system data, set system data.
  - get and set process, file, or device attributes.
- **Communications.**
  - create, delete communication connection.
  - send, receive messages if **message passing model** to **host name** or **process name.**
    - From **client** to **server (open connection & transmit message).**
  - **Shared-memory model** create and gain access to memory regions.
  - transfer status information.
  - attach and detach remote devices.

Types of System Calls (Cont.)

- **Protection.**
  - Control access to resources.
  - Get and set permissions.
  - Allow and deny user access.

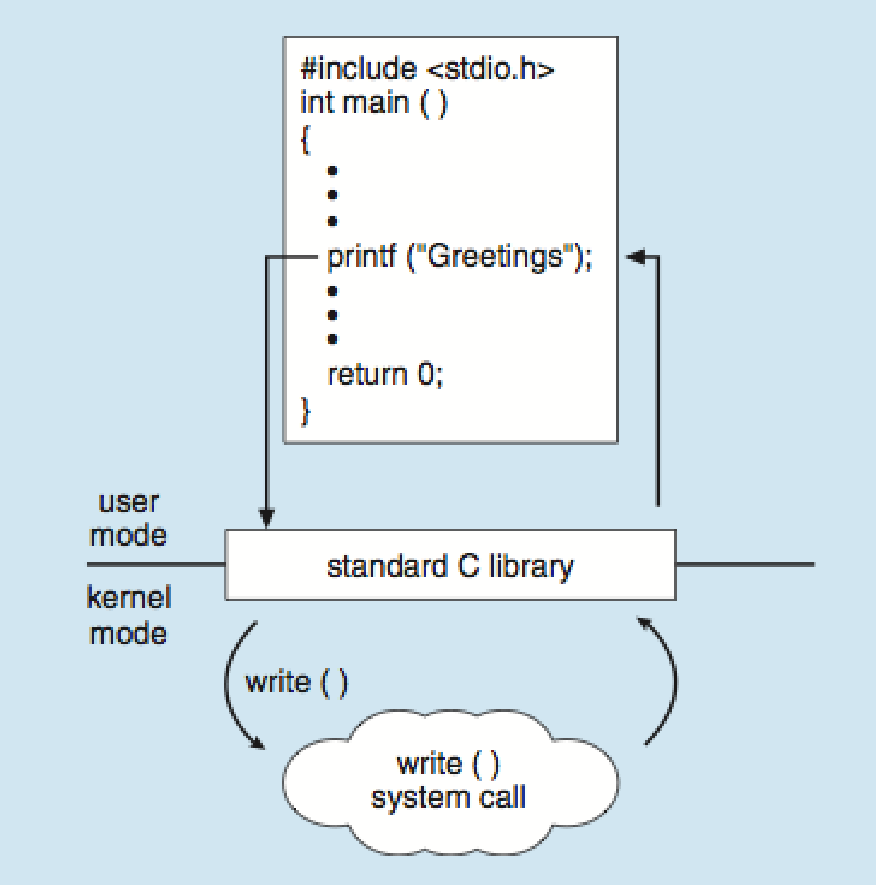
Examples of Windows and Unix System Calls



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

Standard C Library Example

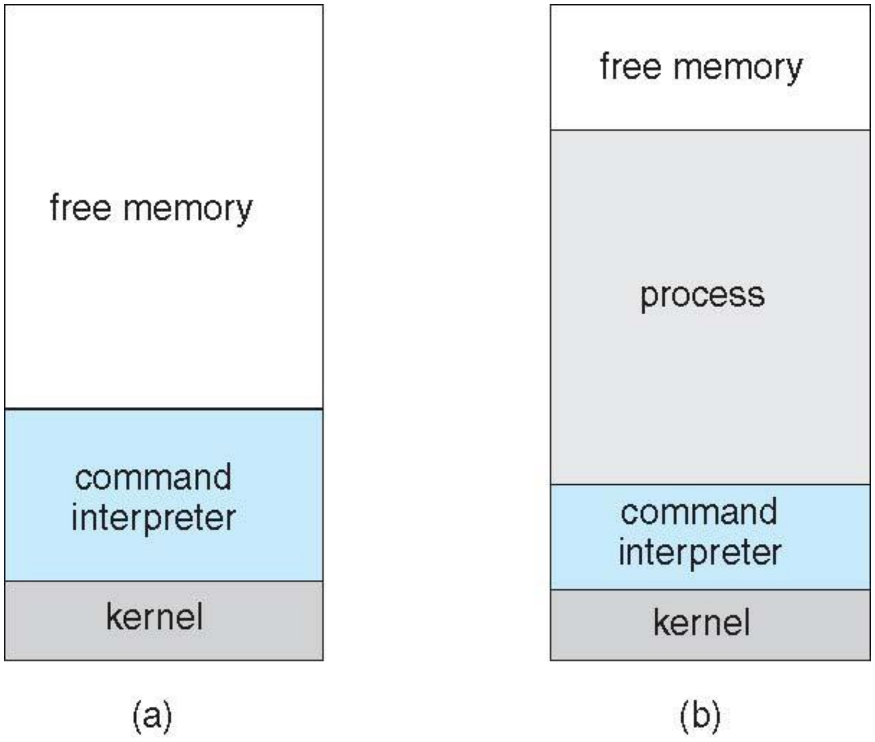
- C program invoking printf() library call, which calls write() system call



```
#include <stdio.h>
int main ( )
{
   •
   •
   •
   printf ("Greetings");
   •
   •
   •
   return 0;
}
```

Process Control example: MS-DOS

- Is a single-tasking OS: one process at a time runs.
- Shell (CI) invoked when system booted.
- Loads program into memory, may overwrite part of the shell. Is this okay?
  - Shell backup on disk.
- Instruction pointer -> 1st instruction in program.
- Program exit -> shell reloaded.
- Above steps are repeated for each process.

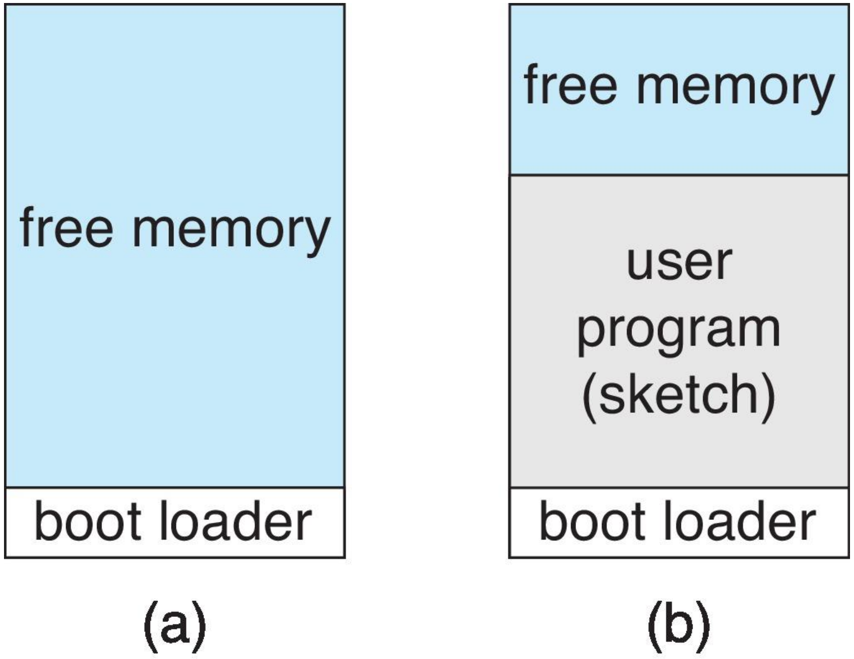


At system startup

running a program

Example: Arduino

- Single-tasking.
- No full fledged operating system.
- Programs (sketches) loaded via USB into flash memory.
- Single memory space.
- Boot loader loads program.
- Program exit -> load another sketch.
- No multitasking.



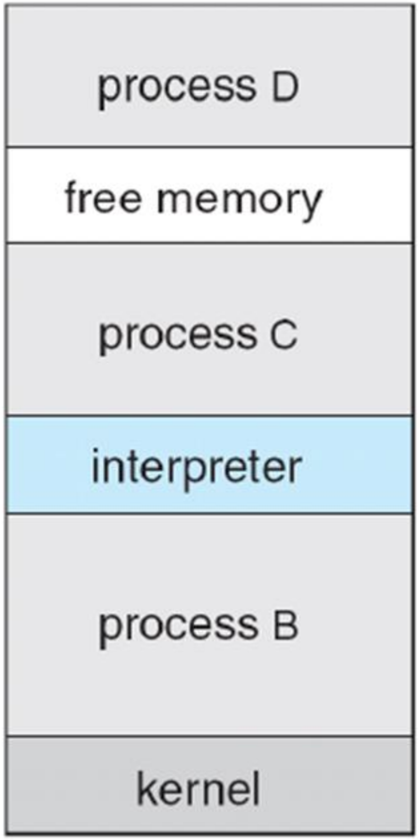
At system startup

running a program

Example: FreeBSD

- Unix variant (from Berkeley Software Distribution (BSD).
- This OS supports multitasking.
- User login -> invoke user's choice of shell.
- Shell executes fork( ) system call to create process.
  - Executes exec( ) to load program into process.
  - Shell waits for process to terminate or **continues with other user commands**.
  - Above is possible due to multitasking.
- Process exits with exit ( ):
  - code = 0 means no error
  - code > 0 means error code
  - Codes are returned to invoking process.
- More on fork ( ) and exit ( ) in chapter 3.

FreeBSD



process D

free memory

process C

interpreter

process B

kernel

Operating Systems Design

- **Q: Why do we need the goals of an OS, before we design it?**
- **A: The goals will influence the design of the OS.**
- E.g.: Real-time system: a specific kind of scheduling algorithm (meet all deadlines), specific kind of memory management (avoid virtual memory).
- Non real-time system: another kind of scheduling (as fast as possible) & memory management (may want to use virtual memory).
- Design of the OS will need to take into account the characteristics of each computing system.
  - Desktop or mobile phone or smart watch.
- Goals may be divided into two categories:
  - **User goals.**
  - **System goals.**

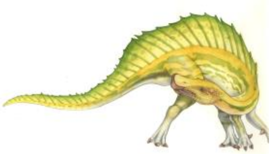
Design Goals

- **User goals?**
  - OS must be easy to use & learn (Windows easier to use than Linux?).
  - Reliability, security.
  - Speed/response time.
- **System goals? (for programmers that design, create & maintain the OS).**
  - OS should be easy to implement (e.g. write code in C).
  - Easy to maintain.
  - Easy to extend functionality.
- An OS is a huge piece of software, with a lot of complexity.
- **Q: How do we tame/control this complexity?**
- **A: We use some principles of software engineering.**

OS Design: Mechanisms & Policies

In general terms:

- **Mechanism: how to do something (procedure, method, technique).**
- **Policy: strategy, guideline.**
- An example from OS: using a timer for specific amount of time for CPU protection.
- **Policy** – timer needs to be how long? 1 millisecond? 1 second?
- **Mechanism** – use a timer for CPU protection.
- **Q: Why bother with policy vs. mechanism?**
- **A: The answer is flexibility.**
  - Policies may change over time.
  - Desirable to have mechanisms that do not change that frequently.
- Example: change the timer duration, from 1 millisecond to 10 milliseconds.
- Easy to implement (*minimum change in code*), mechanism remains the same.
- To change both is more work.



Policy vs. Mechanism

- Another example: have a priority scheme for purpose for process scheduling.
- **This is a policy or a mechanism?**
- **This is a mechanism.**
- Policy 1: give priority to I/O intensive processes.
- Policy 2: give priority to CPU intensive processes.
- Changing policy is not that hard (*min. changes in code*).
- Changing mechanism to go for a new scheduling algorithm is more work.
- **Will see more examples of mechanism and policy throughout the book, be on the lookout**.

Operating Systems Structure

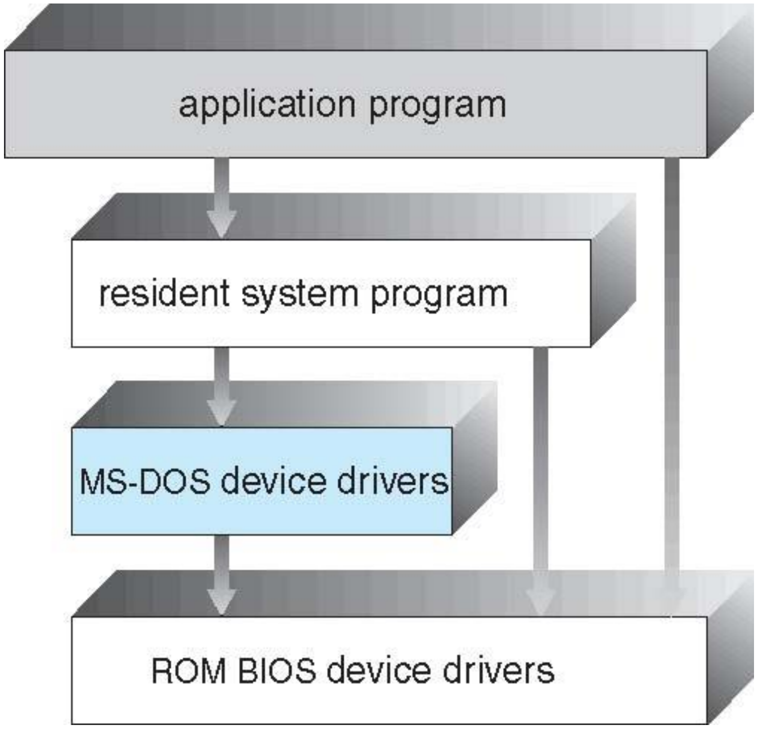
- An analogy:
- While coding programs, two options:
  - A. All code in main ( ) .
  - B. Multiple functions, called from main ( ) .
- Which one do we pick? Why?

Operating System Structure

- General-purpose OS is very large program.
- Millions of lines of code!
- There are various ways to structure an OS.
  - Simple structure : MS-DOS.
  - More complex : UNIX.
  - Layered : an abstraction.
  - Microkernel: Mach.
  - Hybrid: Apple.
- Let’s discuss all of these structures briefly.

Simple Structure -- MS-DOS

- MS-DOS – written to provide the most functionality in the least space.
- BIOS: Basic Input Output System.
- Although MS-DOS has some structure, its interfaces and levels of functionality are not well separated.
- Application program could access I/O devices, such as disks.
- To be fair, it was limited by the hardware of it’s time: Intel 8088.
- No dual mode.



Another Structure -- UNIX

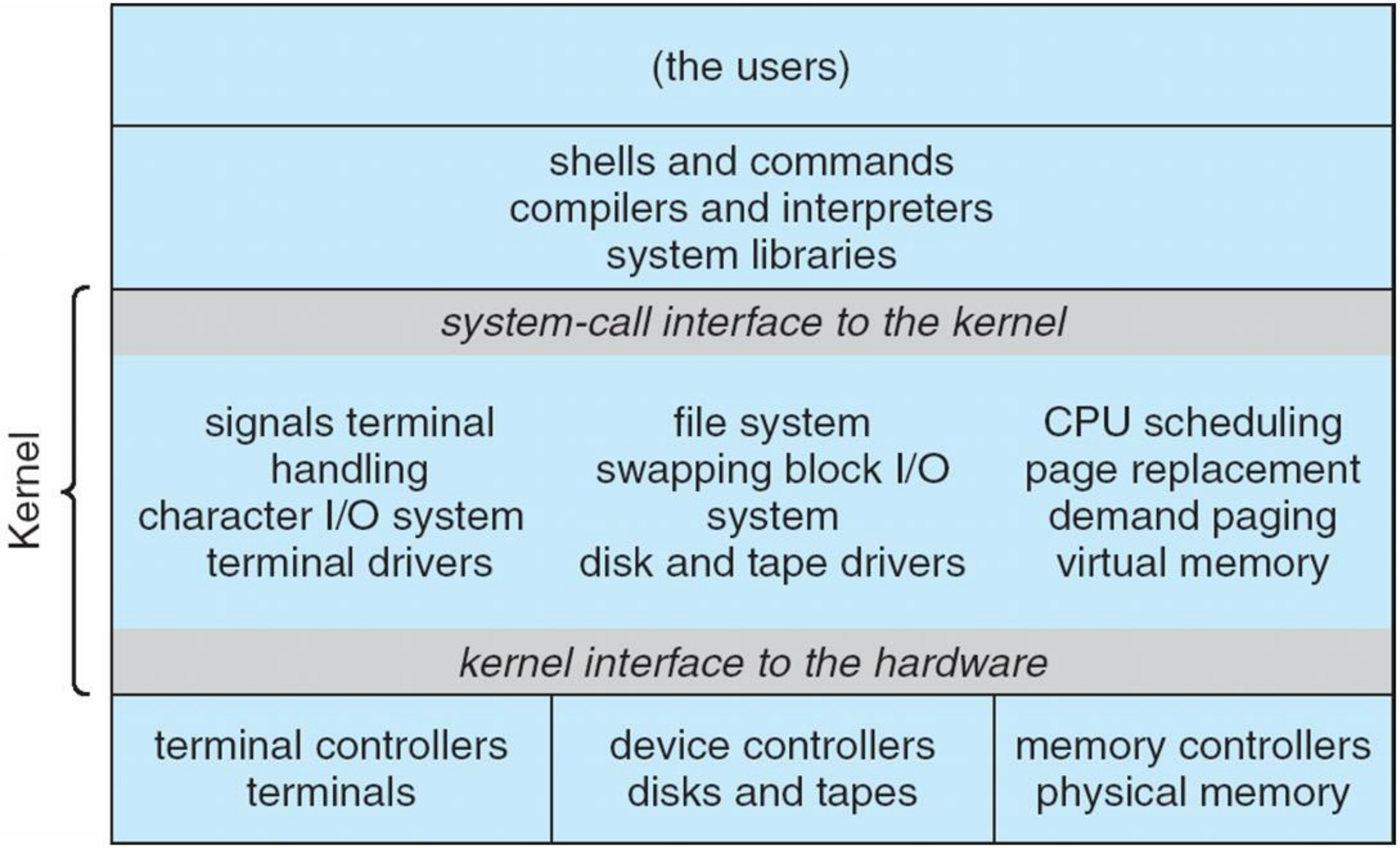
- **Single binary file running in a single address space.**

- Two major parts: (a) the kernel and (b) systems programs.

- The kernel.
  - Consists of everything below the system-call interface and above the physical hardware.
  - Provides the file system, CPU scheduling, memory management, and other operating-system functions (**a large number of functions for one level**).
  - All kernel functionality is in one level. Also called a “monolithic” structure. Disadvantage?
  - Very hard to debug and extend. Too much going on in one block/layer.
    - One part may have interaction with another.
  - Advantage?
  - Fast OS, as various sub parts in same block (same address space), so fast communication.

Traditional UNIX System Structure

Beyond simple but not fully layered.



(the users)

shells and commands
compilers and interpreters
system libraries

system-call interface to the kernel

signals terminal handling
character I/O system
terminal drivers

file system
swapping block I/O system
disk and tape drivers

CPU scheduling
page replacement
demand paging
virtual memory

kernel interface to the hardware

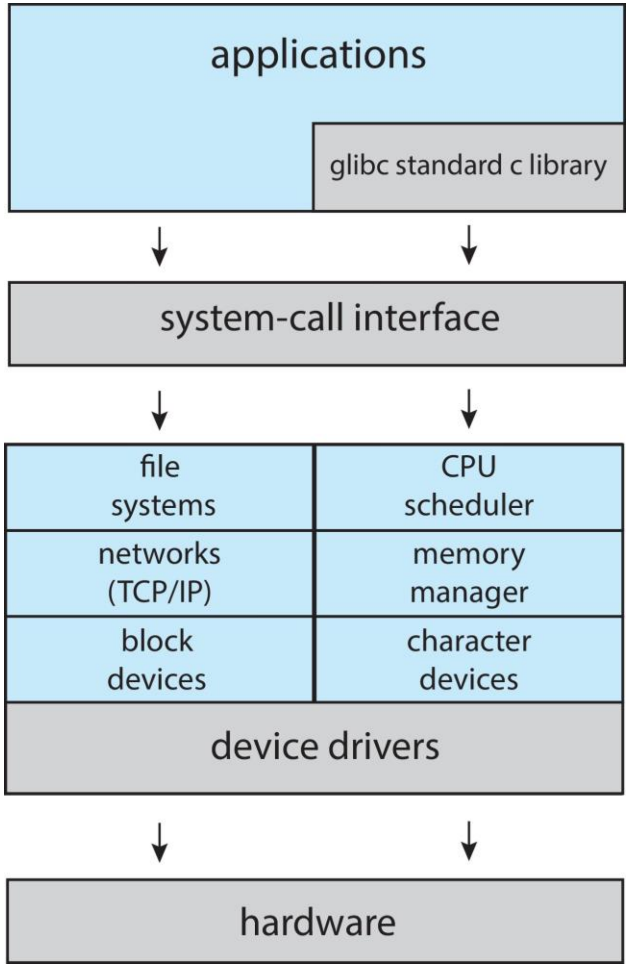
terminal controllers
terminals

device controllers
disks and tapes

memory controllers
physical memory

Linux System Structure

Monolithic plus modular design



applications

glibc standard c library

system-call interface

file systems

CPU scheduler

networks (TCP/IP)

memory manager

block devices

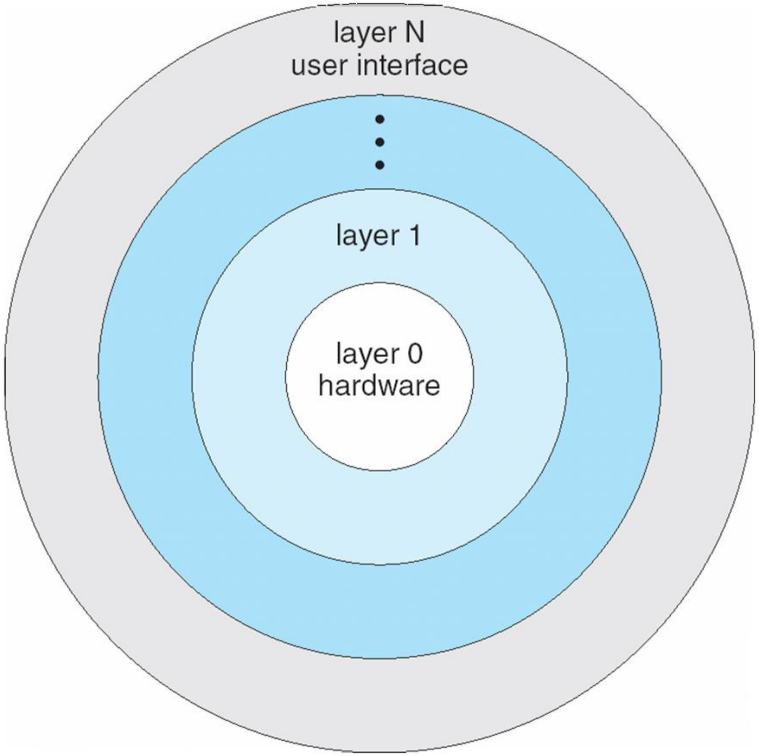
character devices

device drivers

hardware

Layered Approach

- Philosophy: the operating system is divided into a number of layers (levels), each built on top of lower layers.
- The bottom layer (layer 0), is the hardware; the highest (layer N) is the user interface.
- With modularity, layers are selected such that each uses functions (operations) and services of only lower-level layers.
- **Why do this? Advantage?**
- Easier to debug OS/extend it.
  - Can go layer by layer (less complexity).



Layered OS

- Problem with a layered OS?
- **Problem 1: need to define the layers very carefully (can be easy or hard).**
- **Example 1: device driver for disk vs. memory management module.**
- Memory management calls the disk driver, so it has to be at a higher layer.
- **Example 2: device driver for disk vs. CPU scheduler.**
- Disk driver could be above scheduler as it (driver) may have to wait for I/O, during which CPU could be re-scheduled to some other process.
- However, (on large systems) CPU scheduler may have information about all active processes that are too big to fit into memory.
- Some of these need to be swapped in and out of memory from and to a disk.
- In this case, scheduler should be above disk driver.

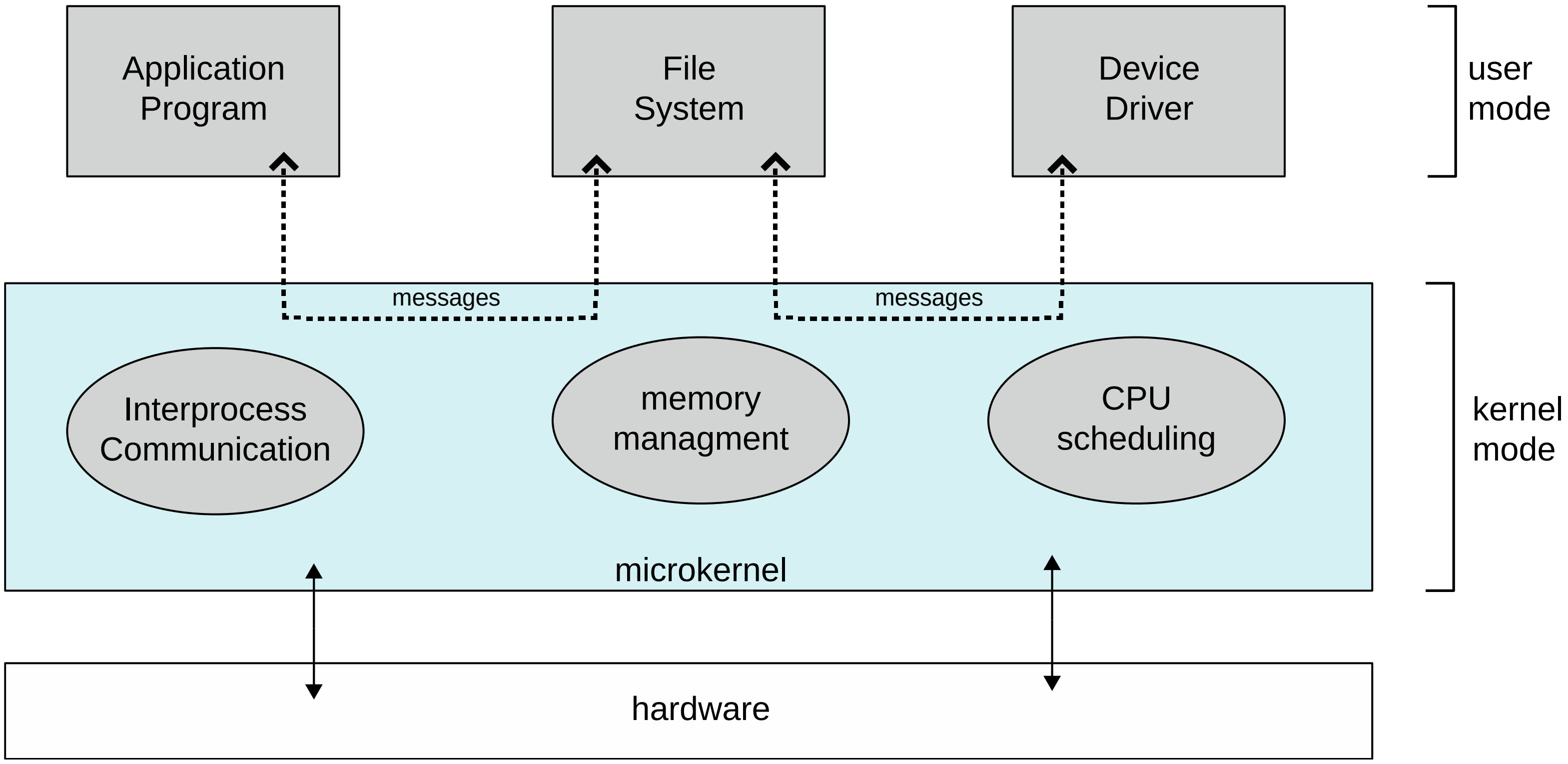
Layered OS

- **Problem 2: layered OS tends to be less efficient.**
- Communicating from one layer to another carries an overhead.
- System call may be modified from one layer to another (parameters modified, data added).
- In case of layered system, enhanced structure & debugging comes at a cost of efficiency.
- Clearly a trade-off is involved between a monolithic OS & a layered OS.
- We will see soon that a lot of OSes go for the **middle path**:
  - Have less layers, with more functionality per layer.

Microkernel

- As UNIX expanded, it became larger & hard to manage.
- Carnegie Mellon University invented the microkernel in the mid 1980s.
- Thus was called Mach.
- **Key idea: OS is structured by removing “non-essential” components from kernel & implementing them in user mode (as user programs).**
- Applications in user space are protected from each other.
- Result is a smaller kernel, providing “core” facilities.
- What to keep in kernel, what to keep in user space → design decision.
- Kernel typically takes care of:
  - CPU scheduling, memory management, inter-process communication (IPC).
- Communication among various modules is through message passing.
- E.g. if a client wishes to access a file, they must interact with the file server.’
- They don’t interact directly, but communicate indirectly by exchanging messages with the microkernel.

Microkernel System Structure



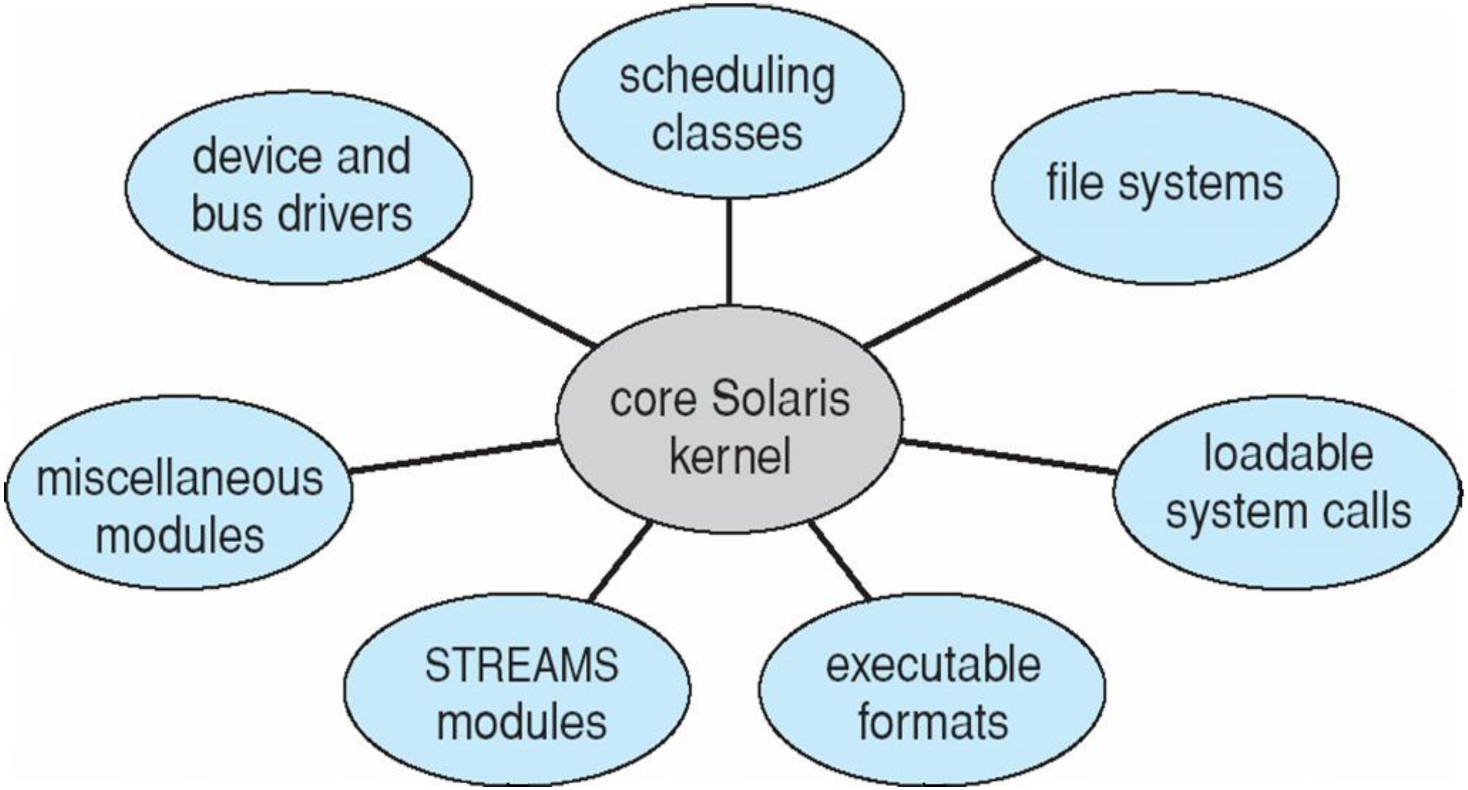
Microkernel System Structure

- **Mach** is an example of a **microkernel**
  - Mac OS X kernel (**Darwin**) based on Mach.
- Benefits of the microkernel approach:
  - Easier to extend the OS. Why?
    - New services added to user space, no change in kernel.
  - Changes to kernel also easier. Why?
  - Less code in kernel.
  - May be more reliable. Why?
    - Services run in user space, less code is running in kernel space.
    - If a service fails, kernel is untouched.
- Disadvantage?
  - Performance overhead of user space to kernel space communication (message passing).

Modules

- Another design approach uses “loadable kernel modules”.
- **(Very small) kernel has a set of core components.**
- **Additional services are linked via modules, during boot time or run-time.**
- This technique used by modern implementations of Linux, Mac, Windows,
- Load the modules as needed, do not load all at once.
- Adding new services becomes easy now.
  - Add another module & load dynamically (as kernel is running), as needed (e.g. when USB drive is inserted).
  - No need to change & recompile the kernel.
- Core components could be:
  - CPU scheduling.
  - Memory management.
- Loadable module could be: file system.

Solaris Modular Approach



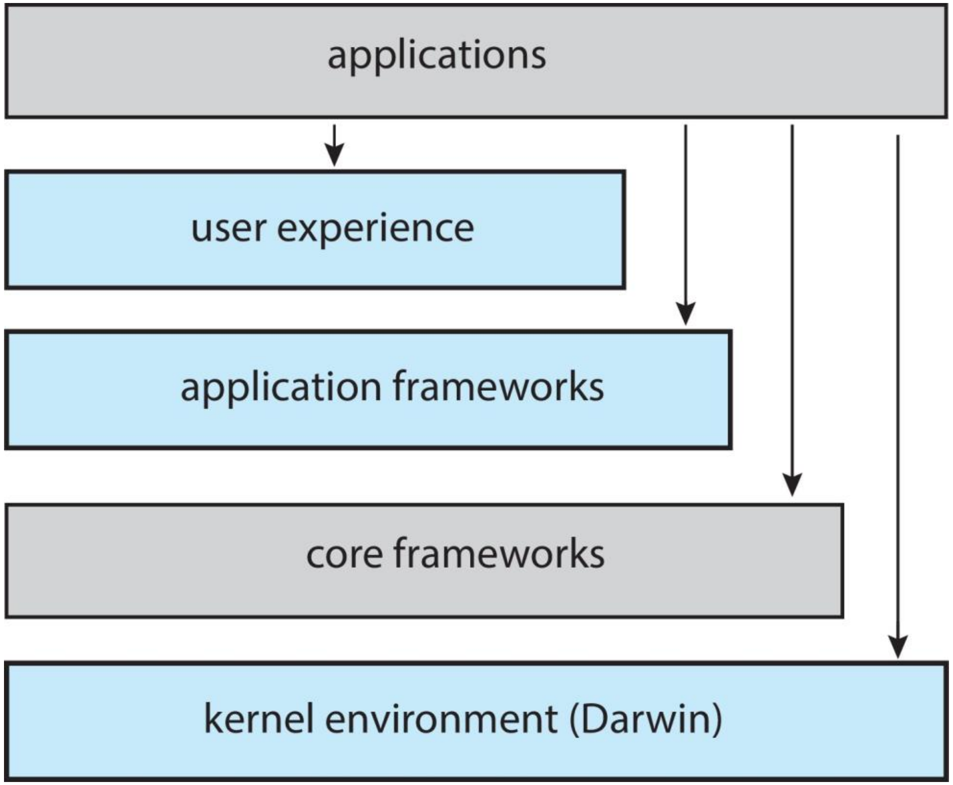
Hybrid Systems

- Most modern operating systems do not follow one ‘pure’ model.
  - Hybrid systems combine multiple approaches.
  - **Can we get the best of both worlds?**
  - Consider “monolithic” vs. “layered” structure.
  - Monolithic system share “same address space”, so are efficient (faster memory accesses).
  - Linux and Solaris kernels are monolithic, plus modular for dynamic loading of functionality (flexibility).
  - Windows mostly monolithic (performance), plus microkernel for different subsystems (called ***personalities)*** and dynamic modules***.***
- We now discuss the structure of three popular Operating Systems.
  - Apple’s OS-X.
  - Apple’s iOS.
  - Google’s Android.

macOS and iOS

- Now, we discuss MacOS and iOS: *some similarities and some differences*.
- **User experience layer**: UI layer. Aqua interface for mouse and trackpad, and Springboard interface for touch.
- **Application framework layer**: Cocoa and cocoa touch frameworks which provide APIs for objective C (macOS) and Swift (iOS) programming languages respectively. These create user applications.
- **Core frameworks:** support graphics APIs (Open GL) and media APIs (Quick Time).
- **Kernel environment**: called Darwin, includes Mach microkernel and BSD UNIX kernel.
- Observation: one can bypass layers. e.g. a C program with no UI that makes POSIX system calls (application directly communicating with the kernel).

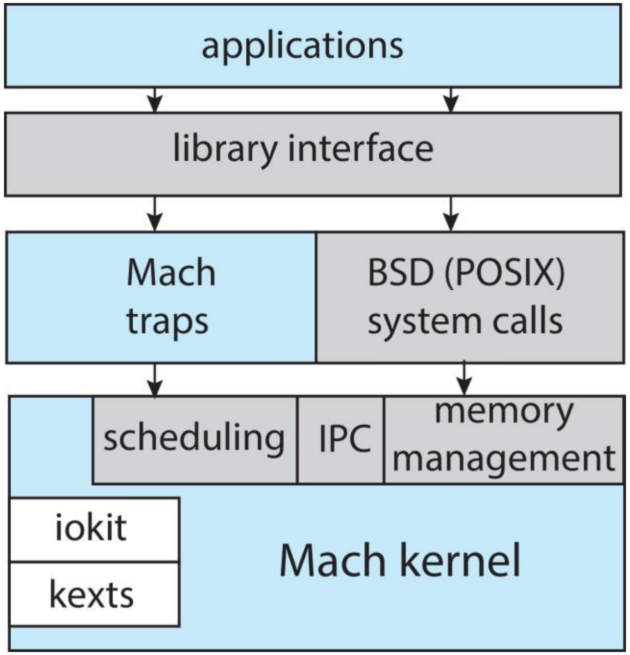
macOS and iOS Structure



Darwin

- We now focus on Darwin = Mach microkernel + BSD UNIX kernel.
- Contains two kernels.
- Hence, two sets of system calls provided: one for BSD & one for Mach.
- Mach provides – memory management, IPC (inter process communication), scheduling.
- BSD provides – command line, support for networking & file systems, implementation of POSIX APIs.
- Below the kernel are:
- I/O kit – an environment for development of device drives.
- Kernel extensions (dynamically loadable modules).
- Darwin has been made open source by Apple.
- Visit: https://opensource.apple.com/

Darwin



applications

library interface

Mach traps

BSD (POSIX) system calls

scheduling

IPC

memory management

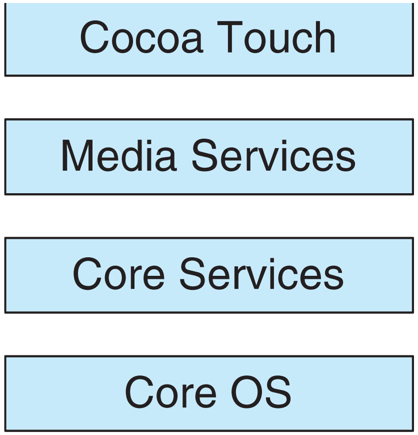
iokit

kexts

Mach kernel

iOS

- Apple mobile OS for ***iPhone***, ***iPad***
  - Structured on Mac OS X, added functionality.
  - Does not run OS X applications natively.
    - Also runs on different CPU architecture (ARM vs. Intel)
  - **Cocoa Touch** Objective-C API for developing apps.
  - **Media services** layer for graphics, audio, video.
  - **Core services** provides cloud computing, databases.
  - Core operating system, based on Mac OS X kernel (closed source).



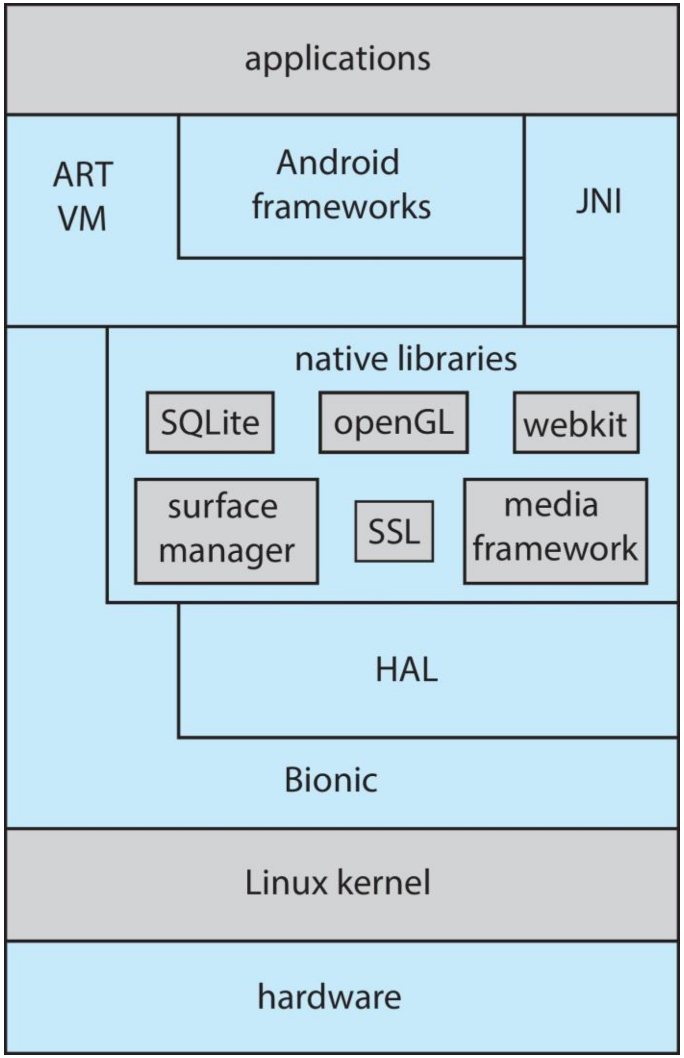
Android

- Developed by Open Handset Alliance (lead by Google).
  - Open Source
- Similar APIs as iOS for graphics, audio.
- Developers create apps in Java Android API: Google developed its own Java API.
- ART VM: virtual machine for Android (apps run (.dex files) in this environment).
- JNI: Java native interface, to write native Java applications (bypass the VM).
- Various libraries are available: webkit for developing web browsers, sqlite for databases, etc.
- Hardware Abstraction Layer (HAL): abstracts underlying hardware, applications get a consistent view.
  - Android runs on heterogeneous hardware.
- Bionic: standard C library for Android – smaller memory footprint, designed for slower GPUs (vs. standard C library glibc).

Android

- Finally, we get to the kernel at the bottom of the architecture.
- Based on Linux kernel but modified.
  - Provides process, memory, device-driver management
  - Adds power management.

Android Architecture



Implementation of an OS

- OS is a large program, written by many people, over a long time.
- Early OSes written in assembly language: e.g. DOS. May download assembly code from - http://www.computerhistory.org/atchm/microsoft-ms-dos-early-source-code/.
- Then programming languages like Algol, PL/1 were used: example is MULTICS (time-sharing OS written in PL/1).
- Now C, C++ are used: Linux.
- Actually, OS is usually a mix of languages:
  - Lowest levels in assembly, for efficiency (device driver code).
  - Most is C code.
  - Systems programs in C, C++, scripting languages like PERL, Python, shell scripts.
- High-level language easier to **code in, compact code.**
  - But, speed is an issue.
- **Emulation** can allow an OS to run on non-native hardware.
- E.g. DOS, written for 8088 assembly can run on an Intel architecture.

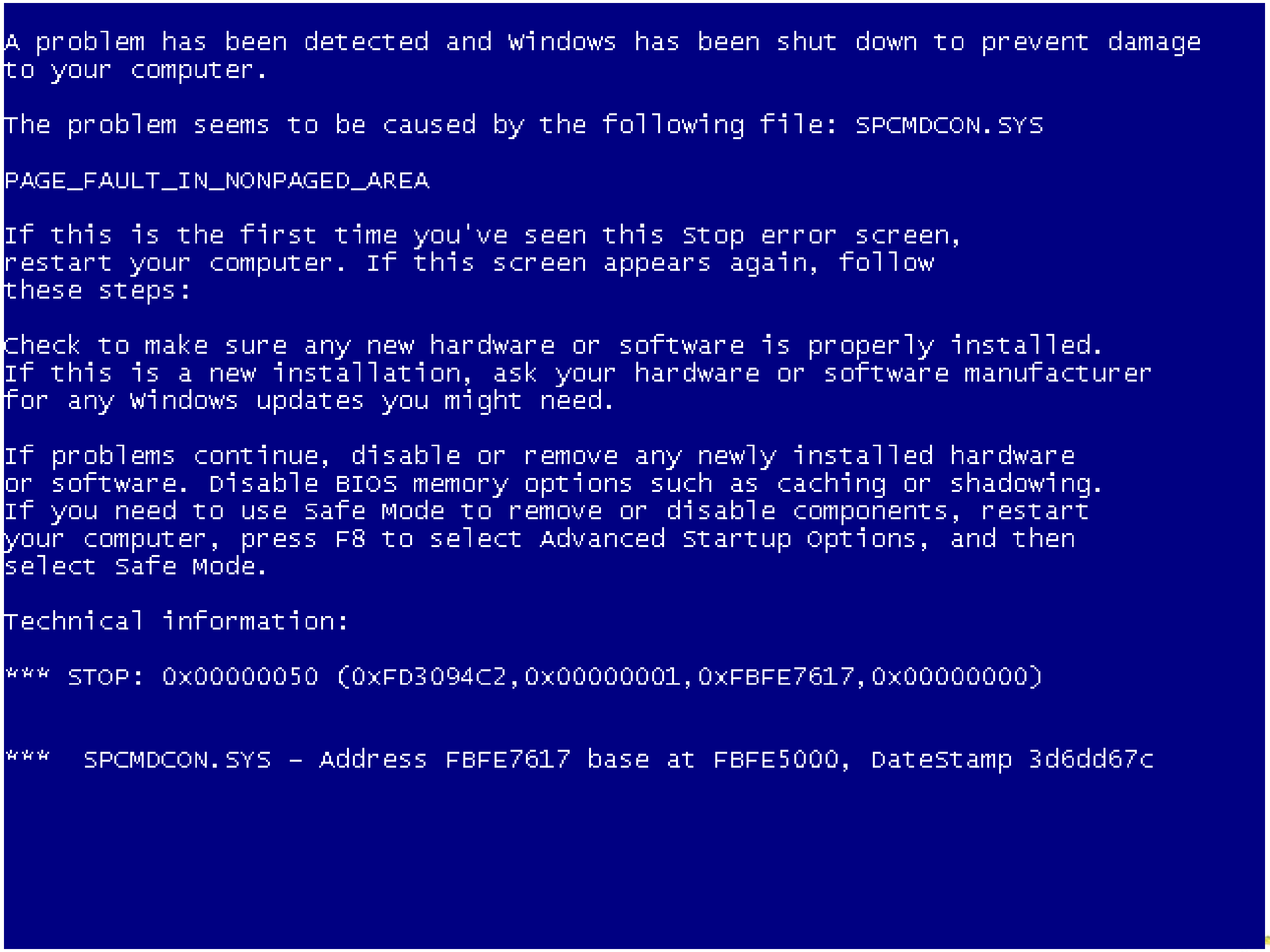
Operating-System Debugging

- **Debugging** is finding and fixing errors or performance problems, both of which are known as **bugs.**
  - May be due to hardware of software.
- OS generate **log files** containing error information for **failed processes**.
  - May be used to correct process code.
- Failure of an application can generate **core** (*old name for memory*) **dump** file capturing memory profile of the process.
  - Info about registers, program counter etc. also dumped.
  - Information may be used to correct process.
- Operating system kernel failure can generate **crash dump** file containing kernel memory.
  - Also known as **kernel panic**.
  - Can look at this file using a debugger to fix the error(s).
  - Causes of crash dump:
    - Faulty hardware – CPU, memory, corrupted disk.
    - Malware.

Some bugs..

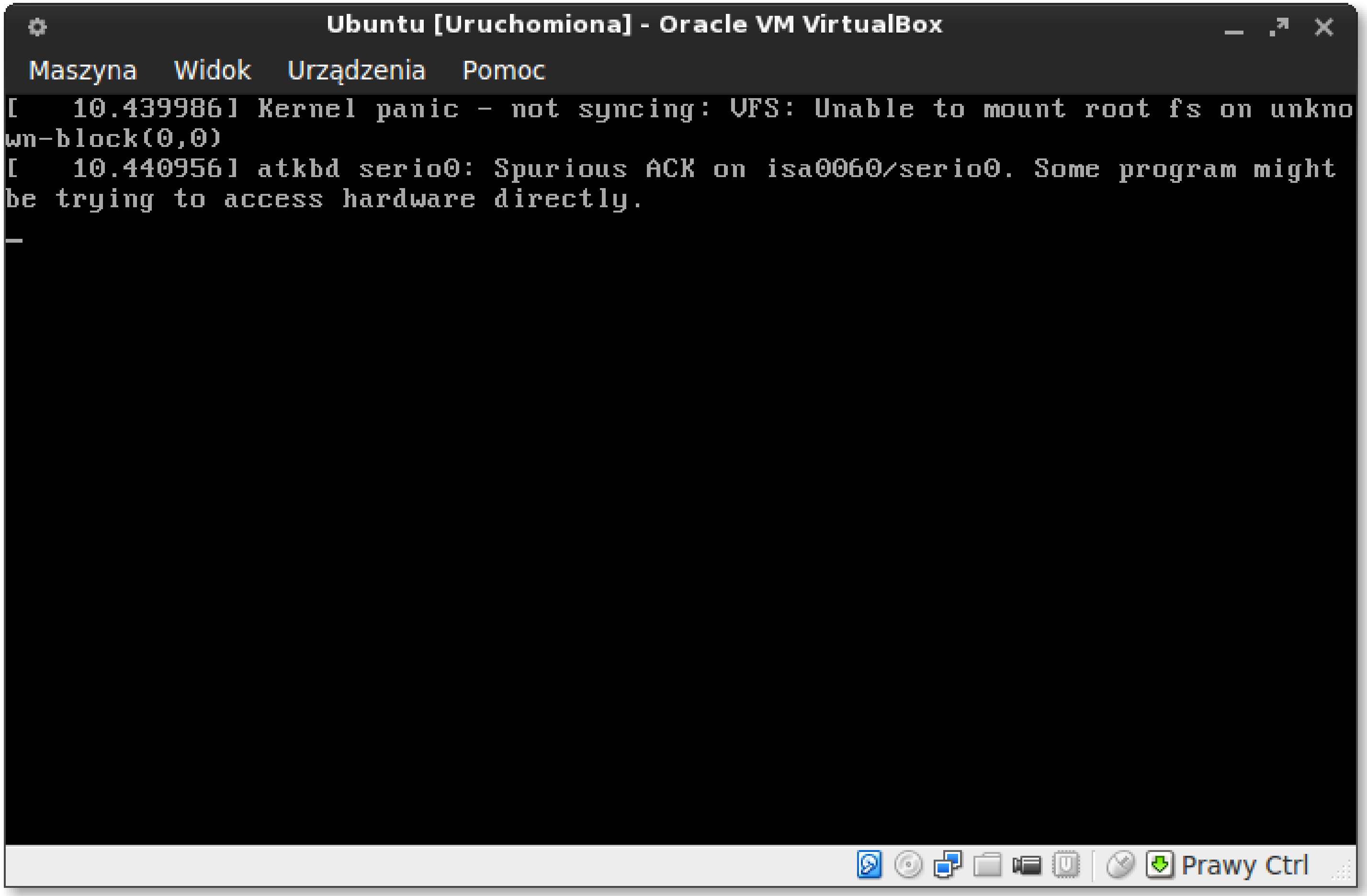
- Some sample bugs in Android:
  - Contacts doubling in texts.
  - Keyboard lag.
  - Web browser hangs.
- Some sample bugs in iOS:
  - Phone bricked when trying to update.
  - Battery drain.
  - Wi-Fi not working.
  - Can’t recognize old device using Bluetooth.

Windows Famous Blue Screen of Death(Source: https://upload.wikimedia.org/wikipedia/commons/a/a8/Windows_XP_BSOD.png)



```
A problem has been detected and windows has been shut down to prevent damage
to your computer.

The problem seems to be caused by the following file: SPCMDCON.SYS

PAGE_FAULT_IN_NONPAGED_AREA

If this is the first time you've seen this Stop error screen,
restart your computer. If this screen appears again, follow
these steps:

Check to make sure any new hardware or software is properly installed.
If this is a new installation, ask your hardware or software manufacturer
for any Windows updates you might need.

If problems continue, disable or remove any newly installed hardware
or software. Disable BIOS memory options such as caching or shadowing.
If you need to use Safe Mode to remove or disable components, restart
your computer, press F8 to select Advanced Startup Options, and then
select Safe Mode.

Technical information:

*** STOP: 0x00000050 (0xFD3094C2,0x00000001,0xFBFE7617,0x00000000)


***  SPCMDCON.SYS - Address FBFE7617 base at FBFE5000, DateStamp 3d6dd67c
```

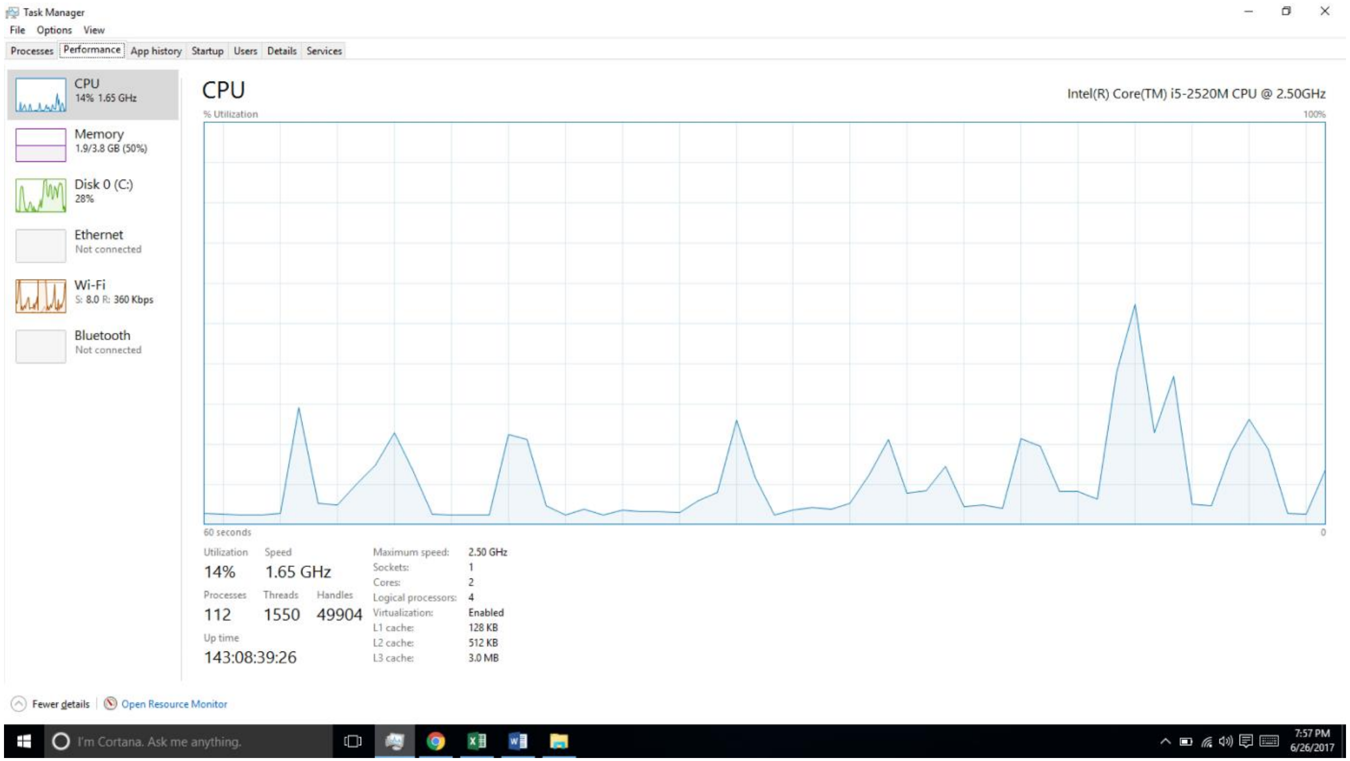
Linux Kernel Panic (Source: https://en.wikipedia.org/wiki/Kernel_panic#/media/File:Ubuntu_13.04_VirtualBox_Kernel_Panic.png)



```
[   10.439986] Kernel panic - not syncing: VFS: Unable to mount root fs on unkno
wn-block(0,0)
[   10.440956] atkbd serio0: Spurious ACK on isa0060/serio0. Some program might
be trying to access hardware directly.
_
```

Performance Tuning

- Improve performance by removing bottlenecks
- OS must provide means of computing and displaying measures of system behavior
- For example, “top” program or Windows Task Manager



CPU 14% 1.65 GHz
Memory 1.9/3.8 GB (50%)
Disk 0 (C:) 28%
Ethernet Not connected
Wi-Fi S: 8.0 R: 360 Kbps
Bluetooth Not connected

CPU Intel(R) Core(TM) i5-2520M CPU @ 2.50GHz

Utilization 14%
Speed 1.65 GHz
Processes 112
Threads 1550
Handles 49904
Up time 143:08:39:26

Maximum speed:	2.50 GHz
Sockets:	1
Cores:	2
Logical processors:	4
Virtualization:	Enabled
L1 cache:	128 KB
L2 cache:	512 KB
L3 cache:	3.0 MB

Performance monitoring tools

- top, htop.
- ps.
- vmstat, iostat, netstat.
- strace.
- bcc.
- A sheet of these tools will be shared with you by email and on Moodle.
- Encouraged to go through and see these tools in action on a Linux machine.
- You may download a Linux VM for this, please see the following for more details: http://people.westminstercollege.edu/faculty/ggagne/osc/vm/index.html (old version) OR
  see: http://people.westminstercollege.edu/faculty/ggagne/osc10e/vm/index.html (new version).

End of Chapter 2