Operating Systems

Chapter 2: Operating-System Structures

Dr. Ahmed Hagag

Scientific Computing Department,
Faculty of Computers and Artificial Intelligence
Benha University

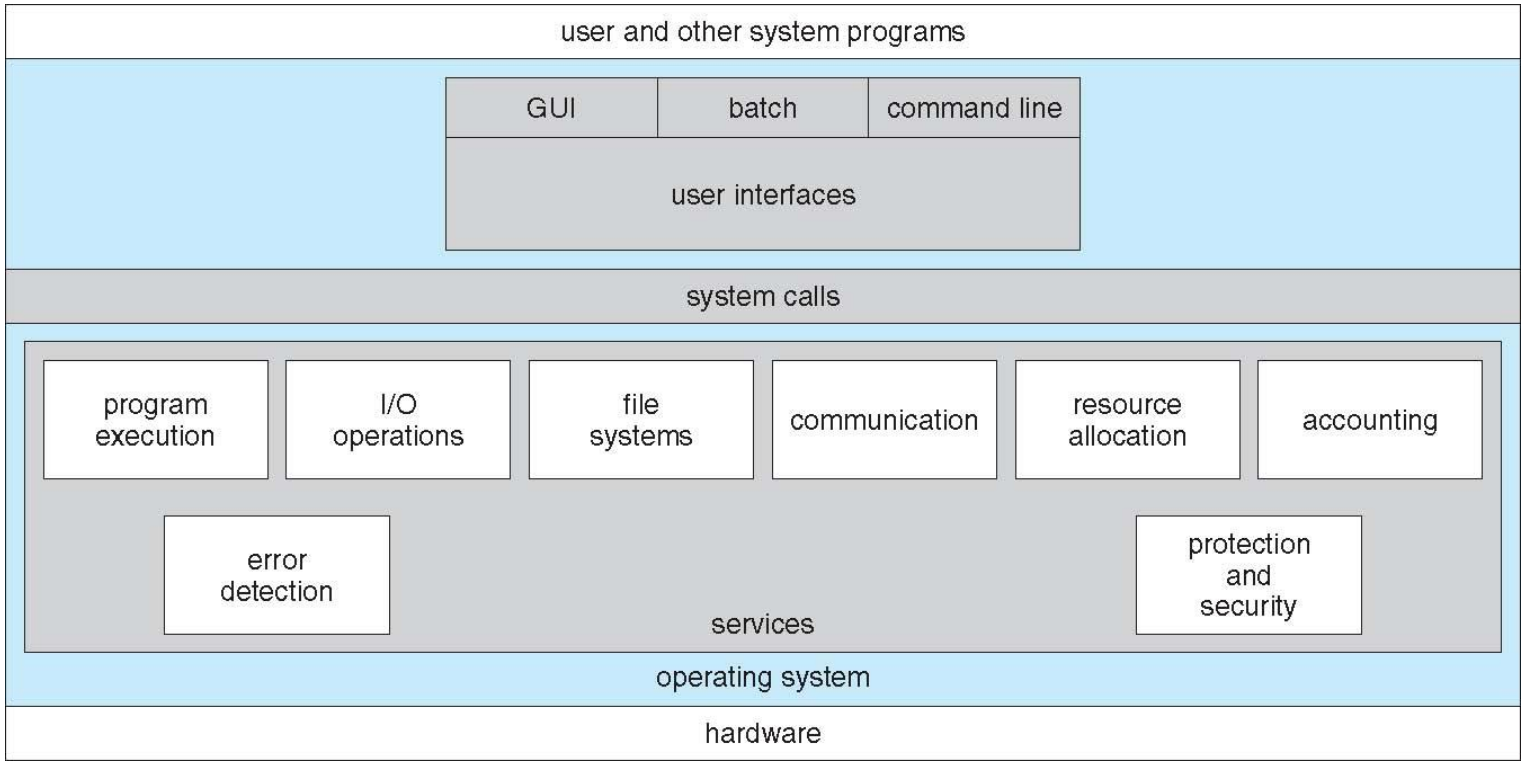
2019

Ch. 2: Operating-System Structures

- Operating System Services
- User Operating System Interface
- System Calls
- Types of System Calls
- System Programs
- Operating System Design and Implementation
- Operating System Structure

Operating System Services (1/4)

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



Operating System Services (2/4)

- One set of operating-system services provides functions that are helpful to the user: (1/3)
  - **User interface** - Almost all operating systems have a user interface (UI).
    - Varies between **Command-Line (CLI)**, **Graphics User Interface (GUI)**, **Batch.**
  - **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.

Operating System Services (2/4)

- One set of operating-system services provides functions that are helpful to the user: (2/3)
  - **File-system manipulation** – The file system is of particular interest. Programs need to read and write files and directories, create and delete them, search them, list file Information, permission management.
  - **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).

Operating System Services (2/4)

- One set of operating-system services provides functions that are helpful to the user: (3/3)
  - **Error detection** – OS needs to be constantly aware of possible errors.
    - May occur in the CPU and memory hardware, in I/O devices, in user program.
    - For each type of error, OS should take the appropriate action to ensure correct and consistent computing.
    - Debugging facilities can greatly enhance the user's and programmer's abilities to efficiently use the system.

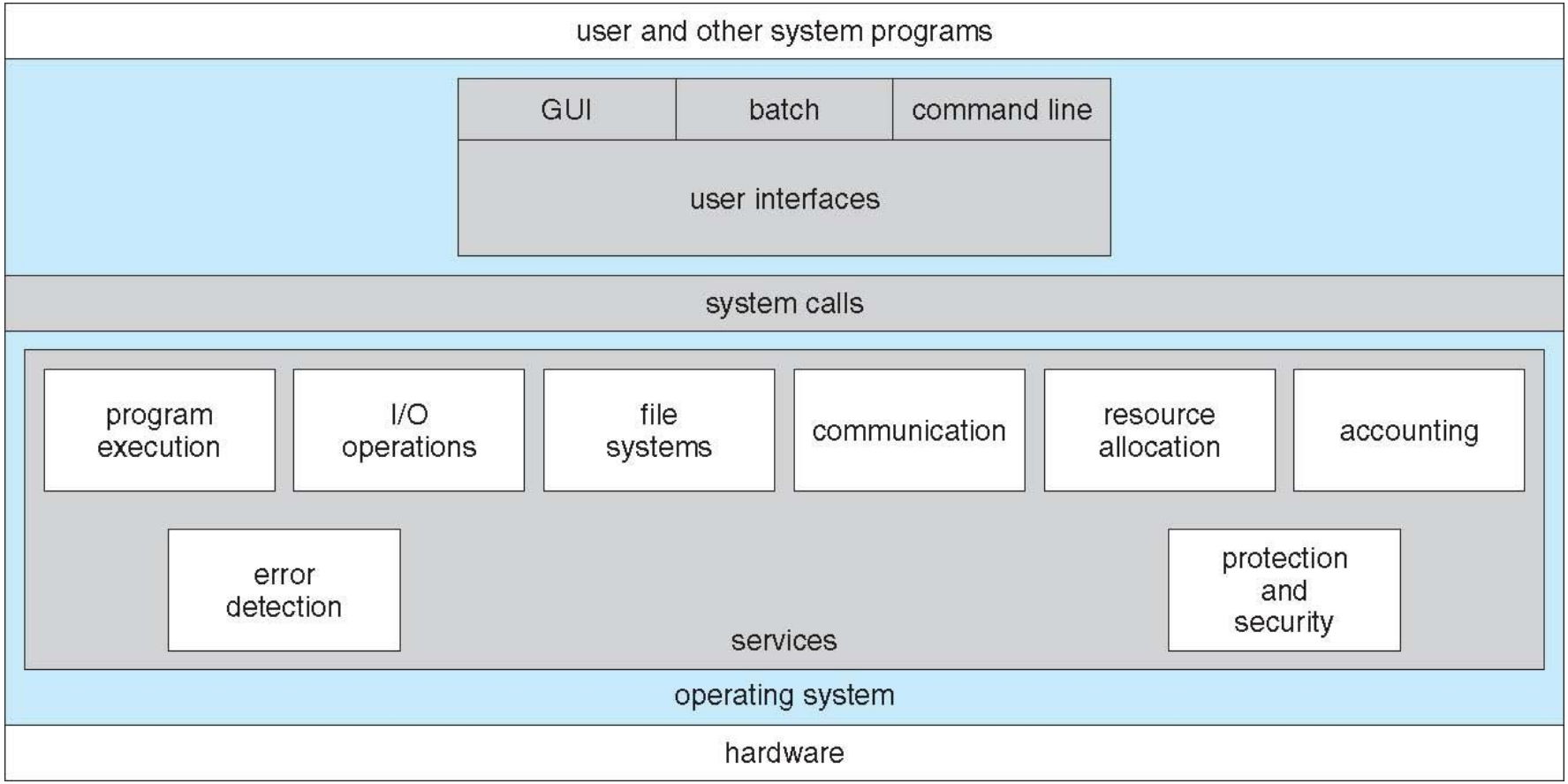
Operating System Services (3/4)

- Another set of OS functions exists for ensuring the efficient operation of the system itself via resource sharing: (1/2)
  - **Resource allocation** - When multiple users or multiple jobs running concurrently, resources must be allocated to each of them.
    - Many types of resources - CPU cycles, main memory, file storage, I/O devices.
  - **Accounting** - To keep track of which users use how much and what kinds of computer resources.

Operating System Services (3/4)

- Another set of OS functions exists for ensuring the efficient operation of the system itself via resource sharing: (2/2)
  - **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.

Operating System Services (4/4)

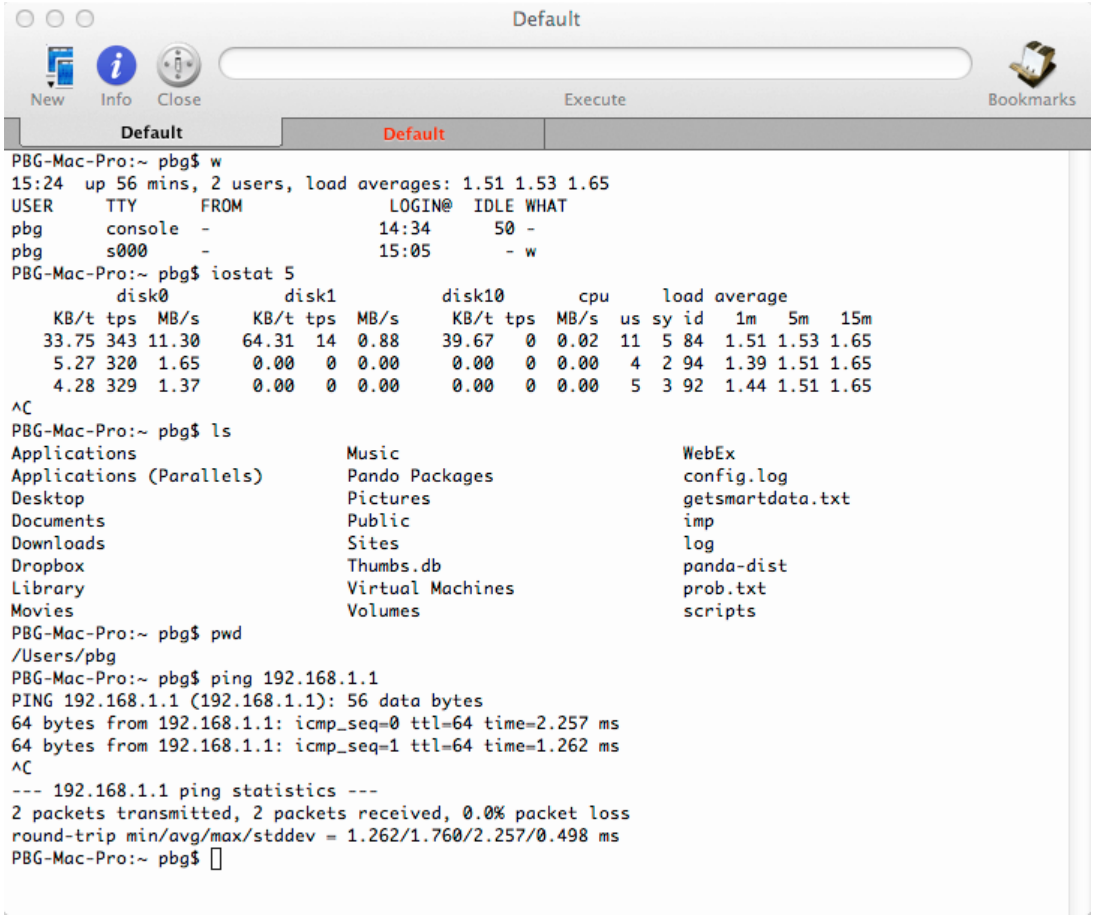


User Operating System Interface (1/5)

- **CLI** or **command interpreter** allows direct command entry.
  - Sometimes implemented in kernel, sometimes by systems program.
  - Sometimes multiple flavors implemented – **shells**
  - Primarily fetches a command from user and executes it.
  - Sometimes commands built-in, sometimes just names of programs.

User Operating System Interface (2/5)

- **Bourne Shell Command Interpreter**




```
PBG-Mac-Pro:~ pbg$ w
15:24  up 56 mins, 2 users, load averages: 1.51 1.53 1.65
USER     TTY      FROM              LOGIN@  IDLE WHAT
pbg      console  -                 14:34      50 -
pbg      s000     -                 15:05       - w
PBG-Mac-Pro:~ pbg$ iostat 5
          disk0           disk1          disk10       cpu     load average
    KB/t tps  MB/s     KB/t tps  MB/s     KB/t tps  MB/s  us sy id   1m   5m   15m
   33.75 343 11.30    64.31  14  0.88    39.67   0  0.02  11  5 84  1.51 1.53 1.65
    5.27 320  1.65     0.00   0  0.00     0.00   0  0.00   4  2 94  1.39 1.51 1.65
    4.28 329  1.37     0.00   0  0.00     0.00   0  0.00   5  3 92  1.44 1.51 1.65
^C
PBG-Mac-Pro:~ pbg$ ls
Applications                    Music                           WebEx
Applications (Parallels)        Pando Packages                  config.log
Desktop                         Pictures                        getsmartdata.txt
Documents                       Public                          imp
Downloads                       Sites                           log
Dropbox                         Thumbs.db                       panda-dist
Library                         Virtual Machines                prob.txt
Movies                          Volumes                         scripts
PBG-Mac-Pro:~ pbg$ pwd
/Users/pbg
PBG-Mac-Pro:~ pbg$ ping 192.168.1.1
PING 192.168.1.1 (192.168.1.1): 56 data bytes
64 bytes from 192.168.1.1: icmp_seq=0 ttl=64 time=2.257 ms
64 bytes from 192.168.1.1: icmp_seq=1 ttl=64 time=1.262 ms
^C
--- 192.168.1.1 ping statistics ---
2 packets transmitted, 2 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 1.262/1.760/2.257/0.498 ms
PBG-Mac-Pro:~ pbg$
```

User Operating System Interface (3/5)

- **Graphics User Interface (GUI)**
- User-friendly **desktop** metaphor interface
  - Usually mouse, keyboard, and monitor.
  - **Icons** represent files, programs, actions, etc.
  - Various mouse buttons over objects in the interface cause various actions (provide information, options, execute function, open directory (known as a **folder**).
- Many systems now include both CLI and GUI interfaces.

User Operating System Interface (4/5)

- **Touchscreen Interfaces**
- Touchscreen devices require new interfaces.
  - Mouse not possible or not desired.
  - Actions and selection based on gestures.
  - Virtual keyboard for text entry.
- Voice commands.

User Operating System Interface (5/5)

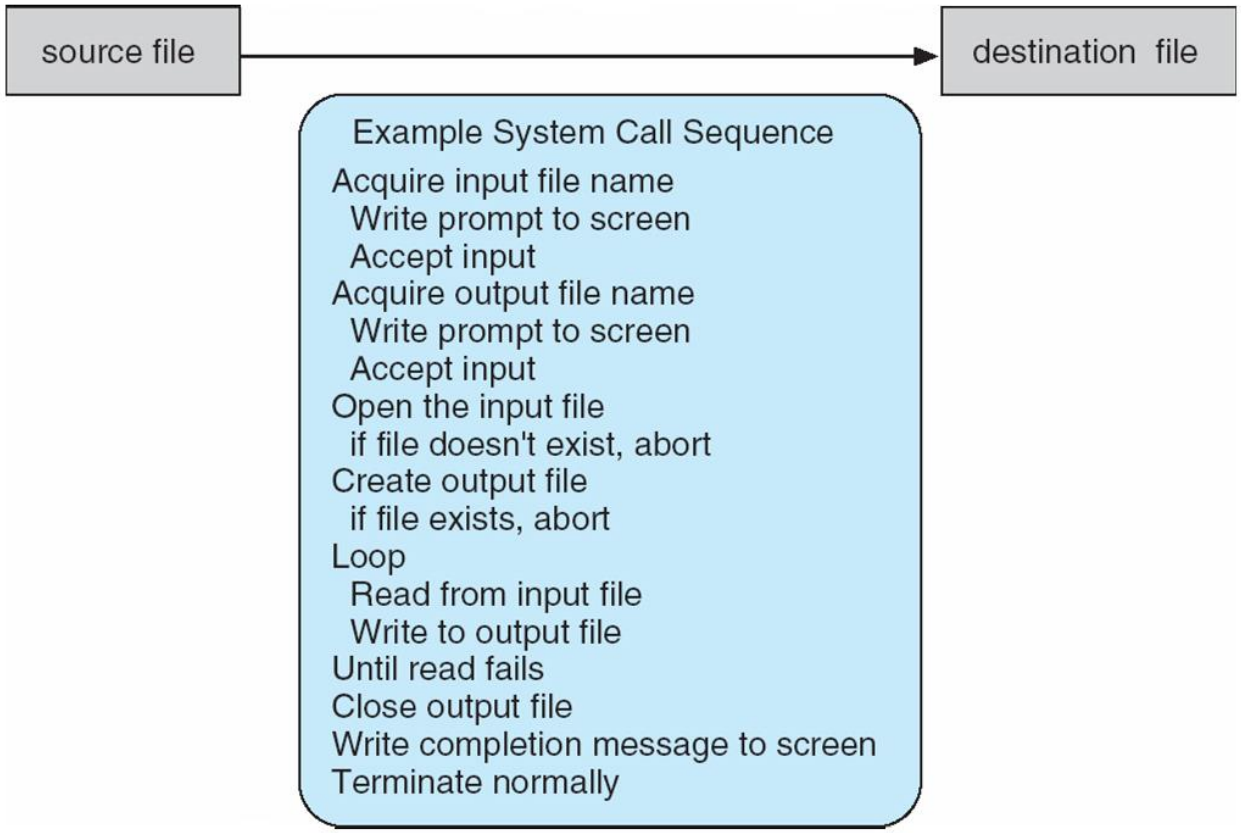
- **The Mac OS X GUI**

System Calls (1/6)

- Programming interface to the **services provided by** the **OS**.
- Typically written in a high-level language (C or C++).
- Mostly accessed by programs via a high-level **Application Programming Interface (API)** rather than direct system call use.
- Three most common APIs are Win32 API for Windows, POSIX API for POSIX-based systems (including virtually all versions of UNIX, Linux, and Mac OS X), and Java API for the Java virtual machine (JVM).

System Calls (2/6)

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



System Calls (3/6)

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)
```

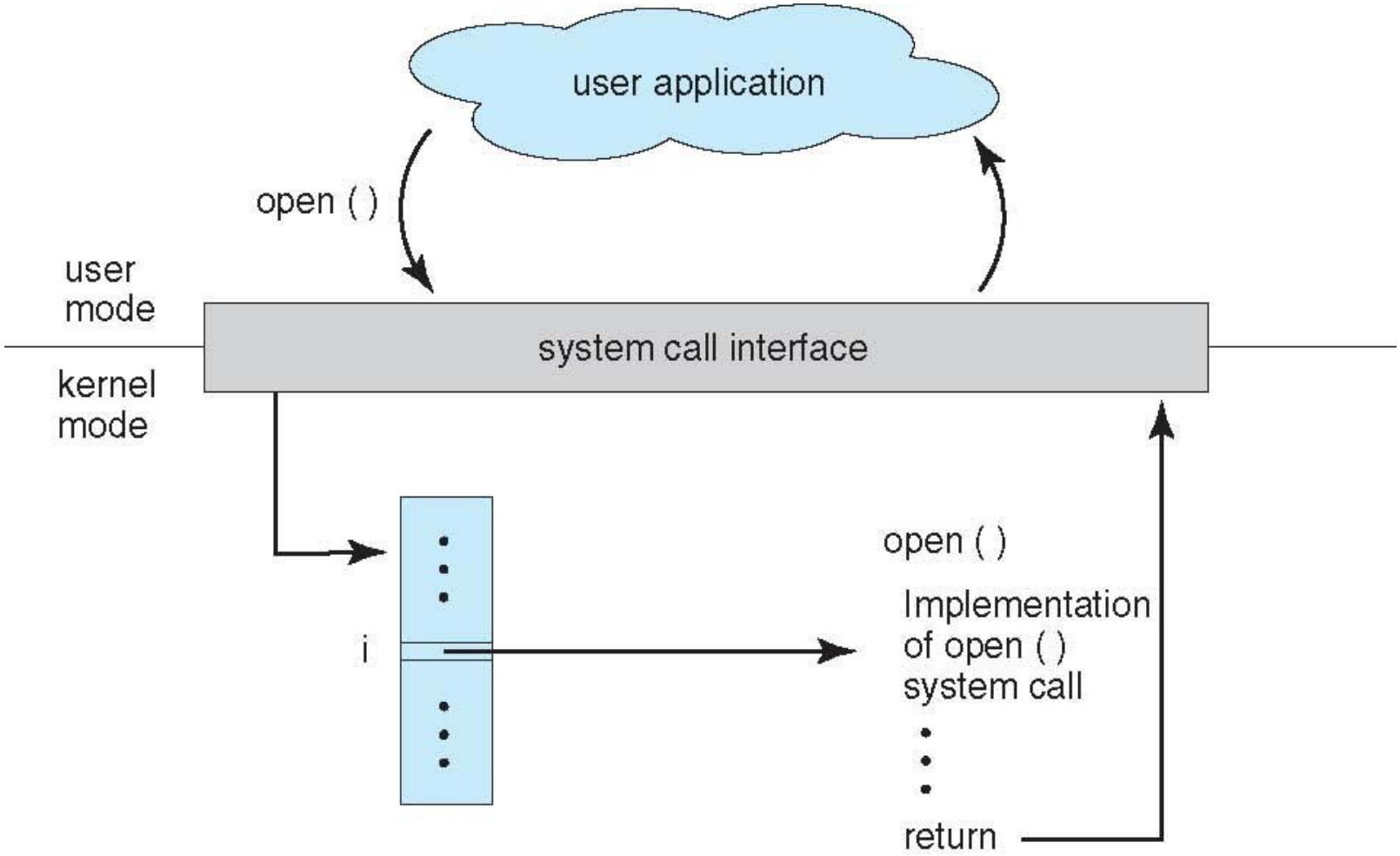
return value — function name — parameters

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 Calls (4/6)

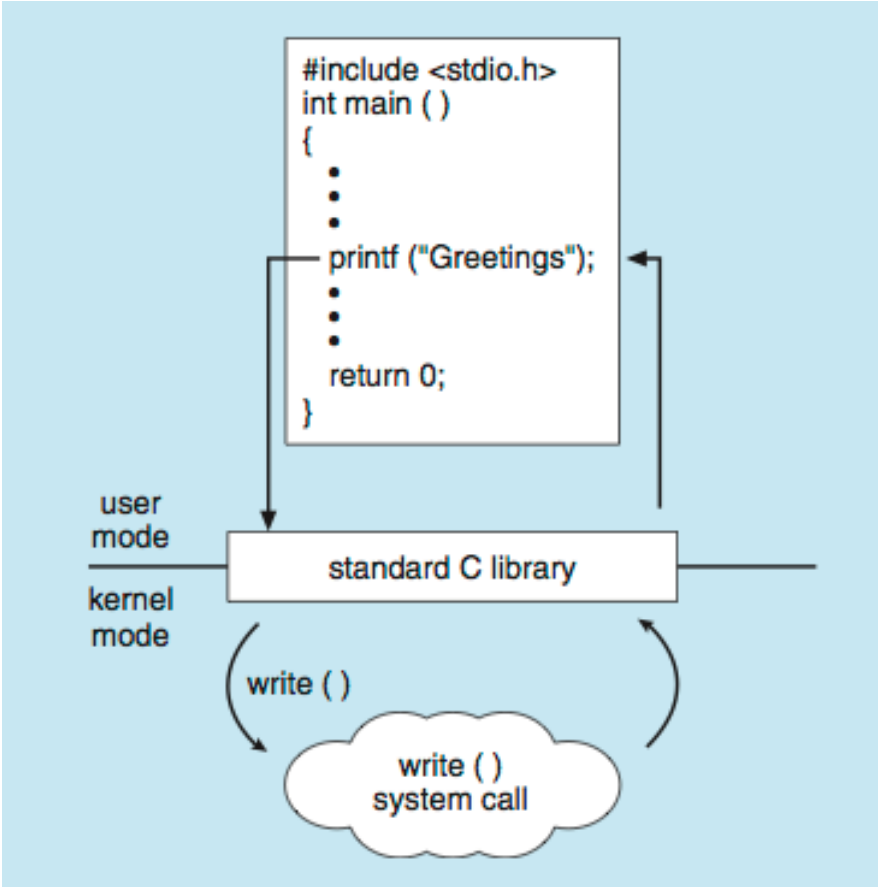


System Calls (5/6)

	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 Calls (6/6)

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



Types of System Calls (1/6)

1. Process control

- create process, terminate process
- end, abort
- load, execute
- get process attributes, set process attributes
- wait for time
- wait event, signal event
- allocate and free memory
- Debugger for determining bugs, single step execution
- Locks for managing access to shared data between processes

Types of System Calls (2/6)

2. File management

- create file, delete file
- open, close file
- read, write, reposition
- get and set file attributes

Types of System Calls (3/6)

3. Device management

- request device, release device
- read, write, reposition
- get device attributes, set device attributes
- logically attach or detach devices

Types of System Calls (4/6)

4. Information maintenance

- get time or date, set time or date
- get system data, set system data
- get and set process, file, or device attributes

Types of System Calls (5/6)

5. Communications

- create, delete communication connection
- send, receive messages if message passing model to host name or process name
- From client to server
- Shared-memory model create and gain access to memory regions
- transfer status information
- attach and detach remote devices

Types of System Calls (6/6)

6. Protection

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

System Programs (1/5)

- System programs provide a convenient environment for program development and execution.
- They can be divided into:
  - File manipulation
  - Status information sometimes stored in a File modification
  - Programming language support
  - Program loading and execution
  - Communications
  - Background services
  - Application programs

System Programs (2/5)

- **File management** - Create, delete, copy, rename, print, dump, list, and generally manipulate files and directories.
- **Status information**
  - Some ask the system for info - date, time, amount of available memory, disk space, number of users.
  - Others provide detailed performance, logging, and debugging information.
  - Typically, these programs format and print the output to the terminal or other output devices.

System Programs (3/5)

- **File modification**
  - Text editors to create and modify files.
  - Special commands to search contents of files or perform transformations of the text.
- **Programming-language support** - Compilers, assemblers, debuggers and interpreters sometimes provided.

System Programs (4/5)

- **Communications -** Provide the mechanism for creating virtual connections among processes, users, and computer systems.
  - Allow users to send messages to one another's screens, browse web pages, send electronic-mail messages, log in remotely, transfer files from one machine to another.

System Programs (5/5)

- **Background Services**
  - Launch at boot time.
    - Some for system startup, then terminate.
    - Some from system boot to shutdown.
  - Provide facilities like disk checking, process scheduling, error logging, printing.
- **Application programs**
  - Don’t pertain to system.
  - Run by users.
  - Not typically considered part of OS.

OS Design and Implementation (1/3)

- Design and Implementation of OS not "**solvable**", but some approaches have proven successful.
- Internal structure of different Operating Systems can vary widely.
- Start the design by **defining goals** and specifications.
- Affected by choice of hardware, type of system.

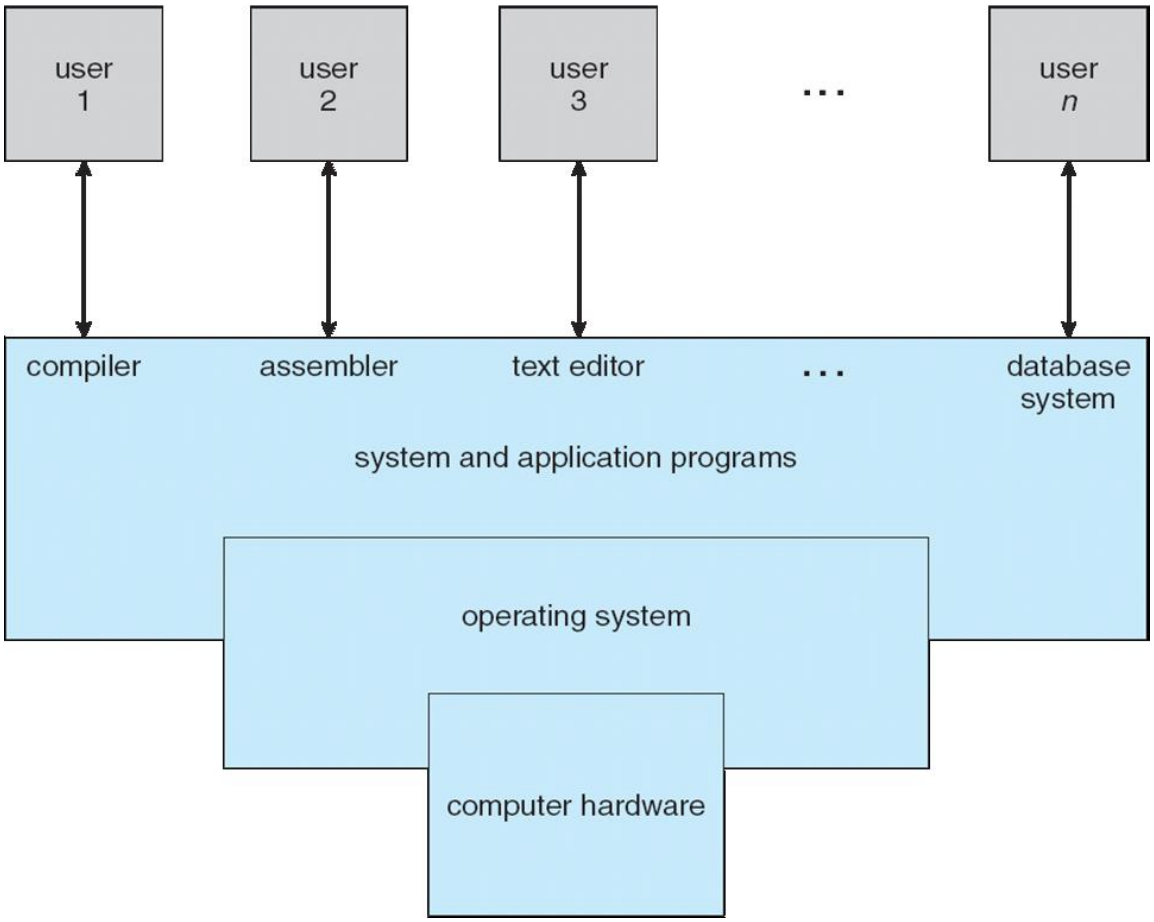
OS Design and Implementation (2/3)

- **User goals and System goals**
  - **User goals** – operating system should be convenient to use, easy to learn, reliable, safe, and fast.
  - **System goals** – operating system should be easy to design, implement, and maintain, as well as flexible, reliable, error-free, and efficient.
- Specifying and designing an OS is highly creative task of software engineering.

OS Design and Implementation (3/3)

- **Much variation**
  - Early OSes in assembly language.
  - Then system programming languages like Algol, PL/1
  - Now C, C++
- **Actually usually a mix of languages**
  - Lowest levels in assembly
  - Main body in C
  - Systems programs in C, C++, scripting languages like PERL, Python, shell scripts.

Operating System Structure (1/8)

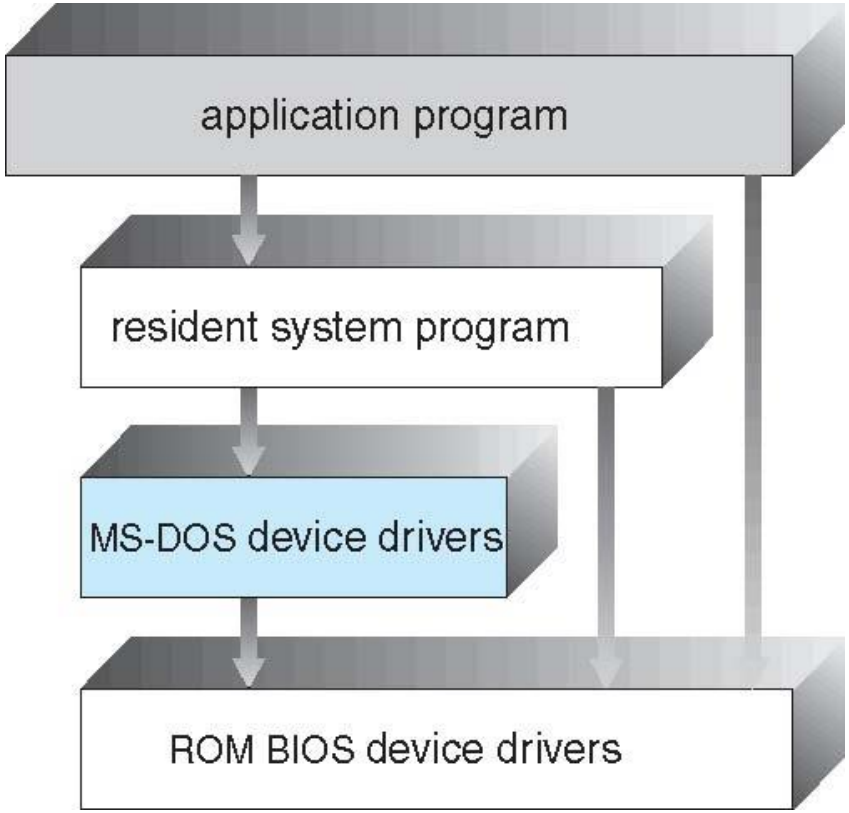


Operating System Structure (2/8)

- General-purpose OS is very large program.
- Various ways to structure ones:
  - Simple structure – MS-DOS
  - More complex – UNIX
  - Layered
  - Microkernel – Mach

Operating System Structure (3/8)

Simple structure – MS-DOS



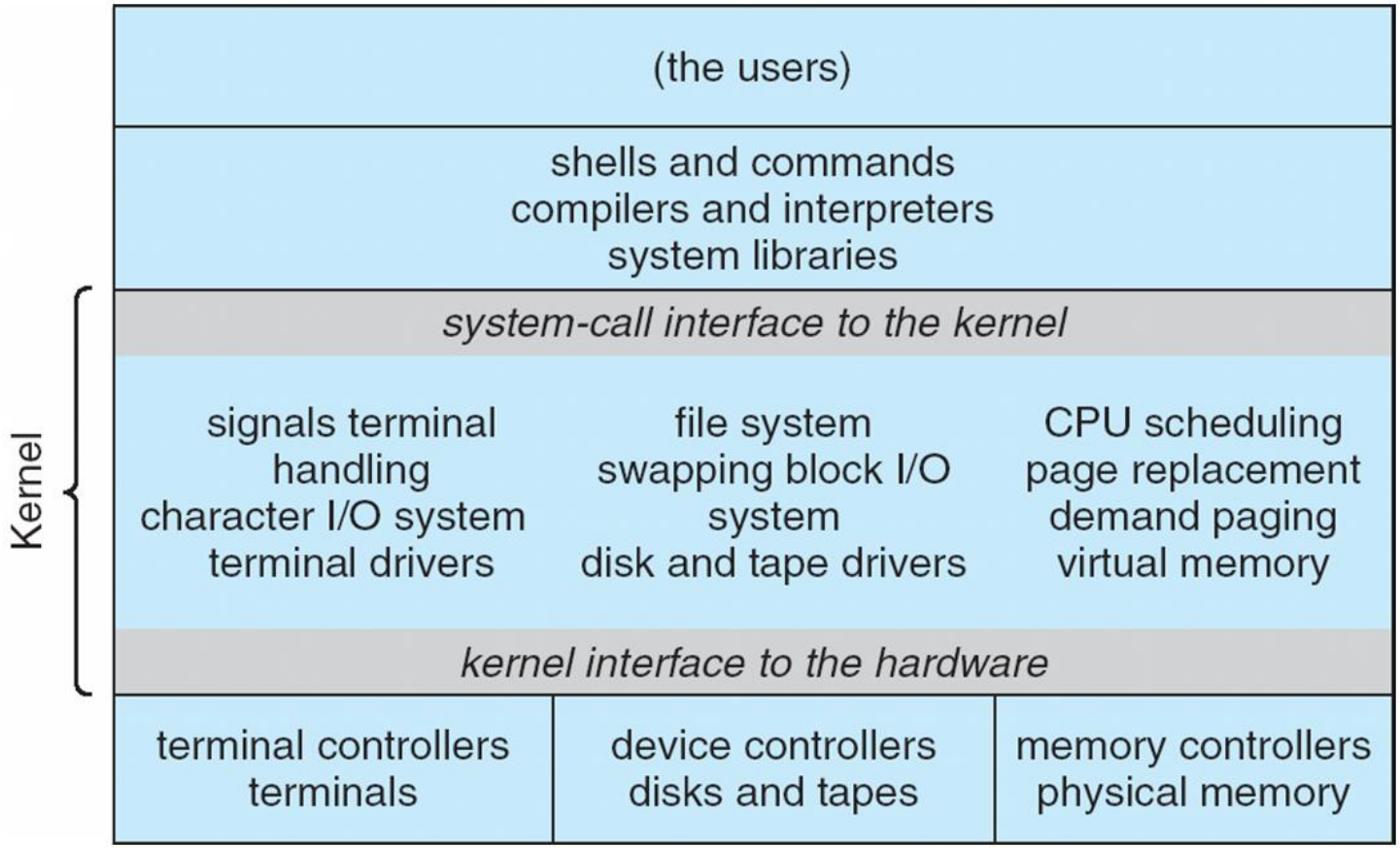
Operating System Structure (4/8)

The UNIX OS consists of two separable parts:

- **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.

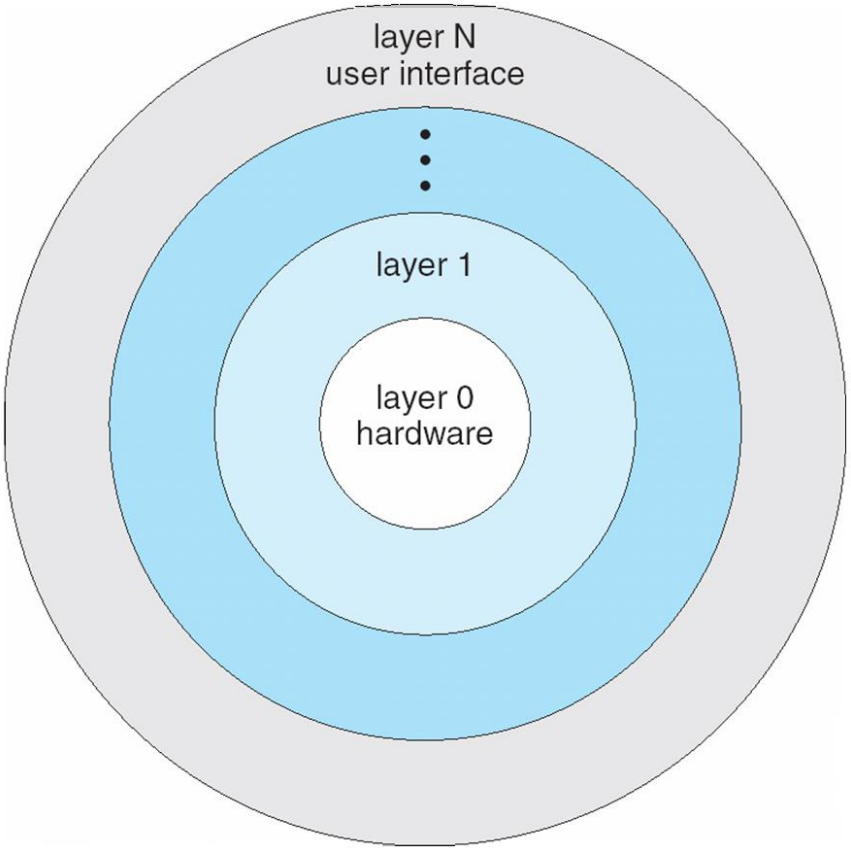
Operating System Structure (5/8)

Traditional UNIX System Structure



Operating System Structure (6/8)

Layered Approach



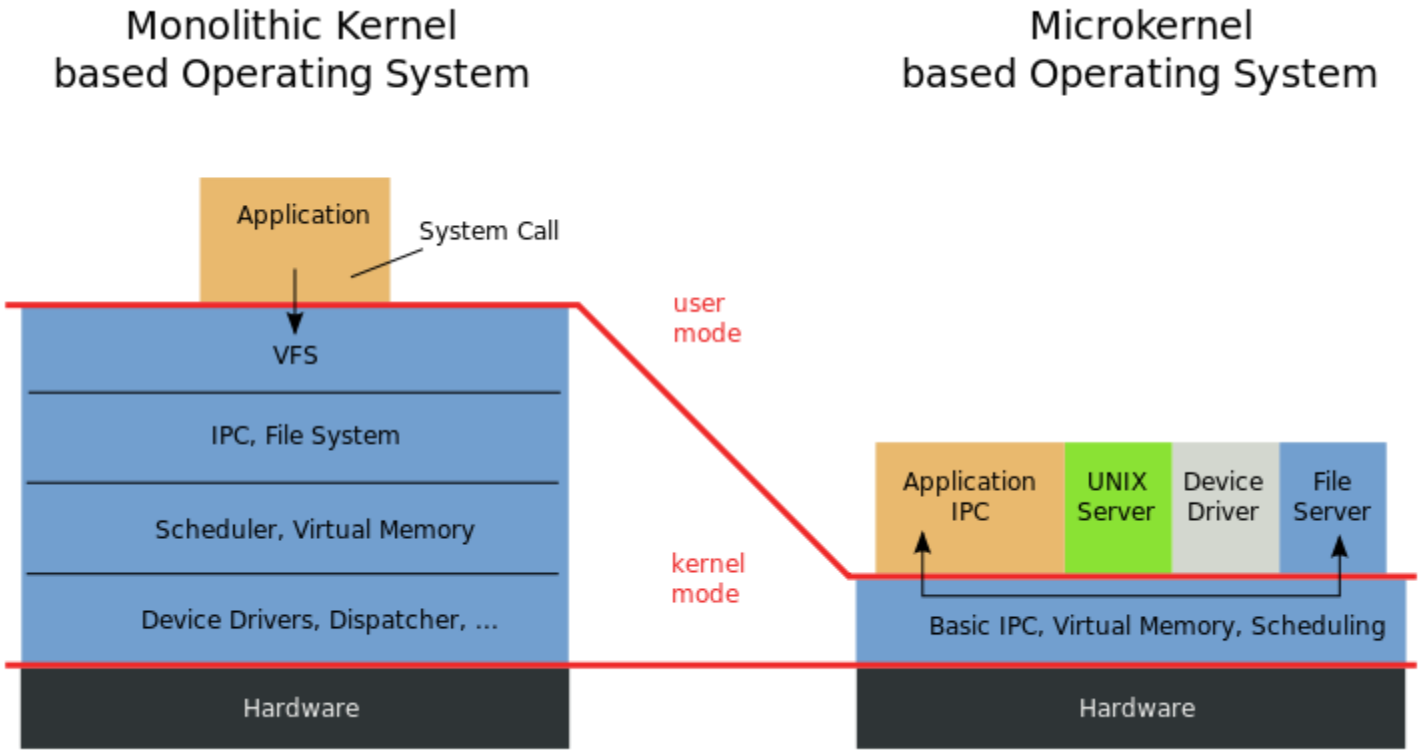
Operating System Structure (7/8)

Microkernel System Structure

- Moves as much from the kernel into user space
- **Mach** example of **microkernel**
- Mac OS X kernel (Darwin) partly based on Mach

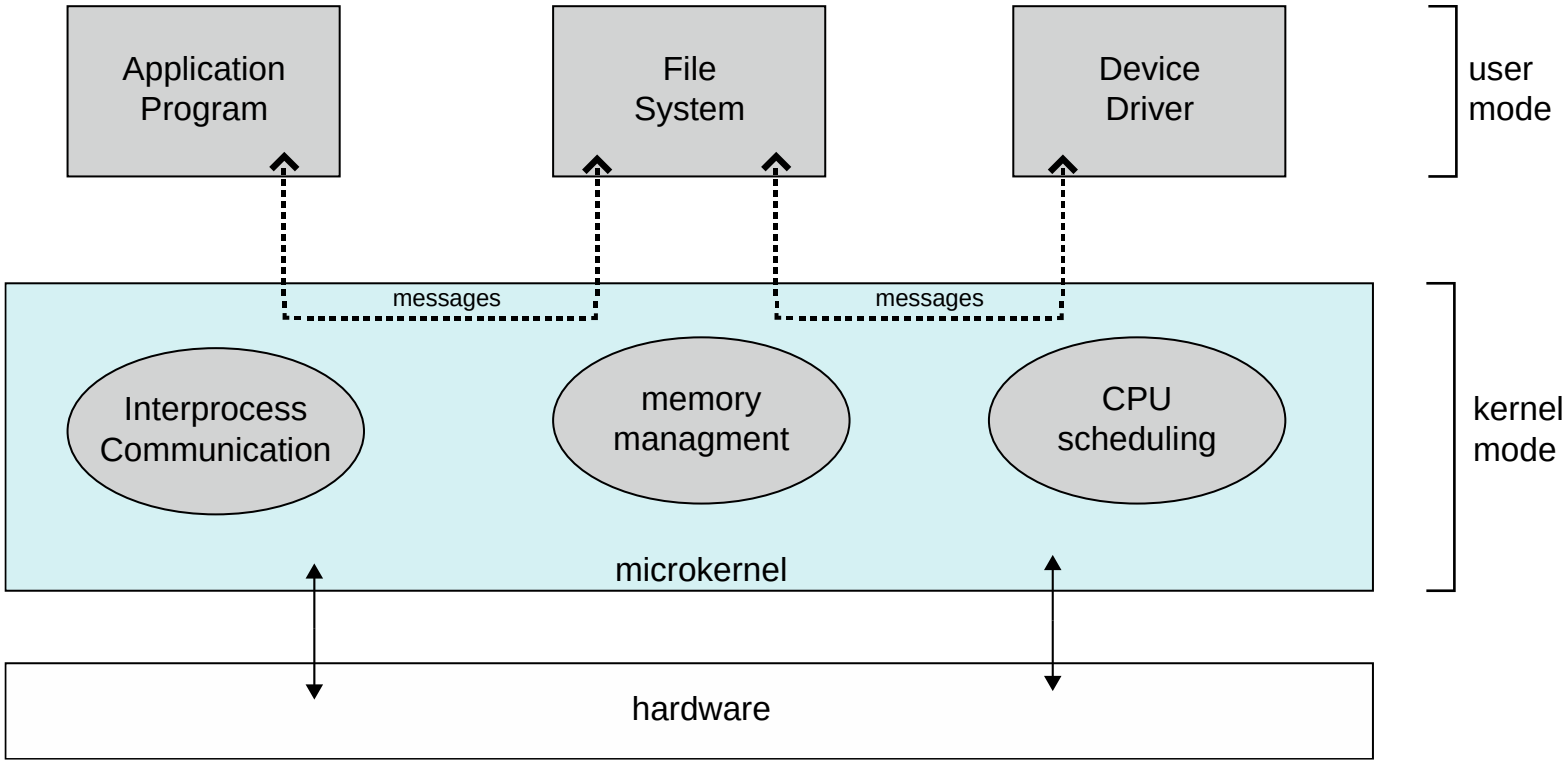
Operating System Structure (7/8)

Microkernel System Structure



Operating System Structure (8/8)

Microkernel System Structure



Thank You

Dr. Ahmed Hagag

ahagag@fci.bu.edu.eg

https://www.youtube.com/channel/UCzYgAyyZTLfnLFjQexOKxbQ

https://www.facebook.com/ahmed.hagag.71/