Operating system structures

Tran, Van Hoai

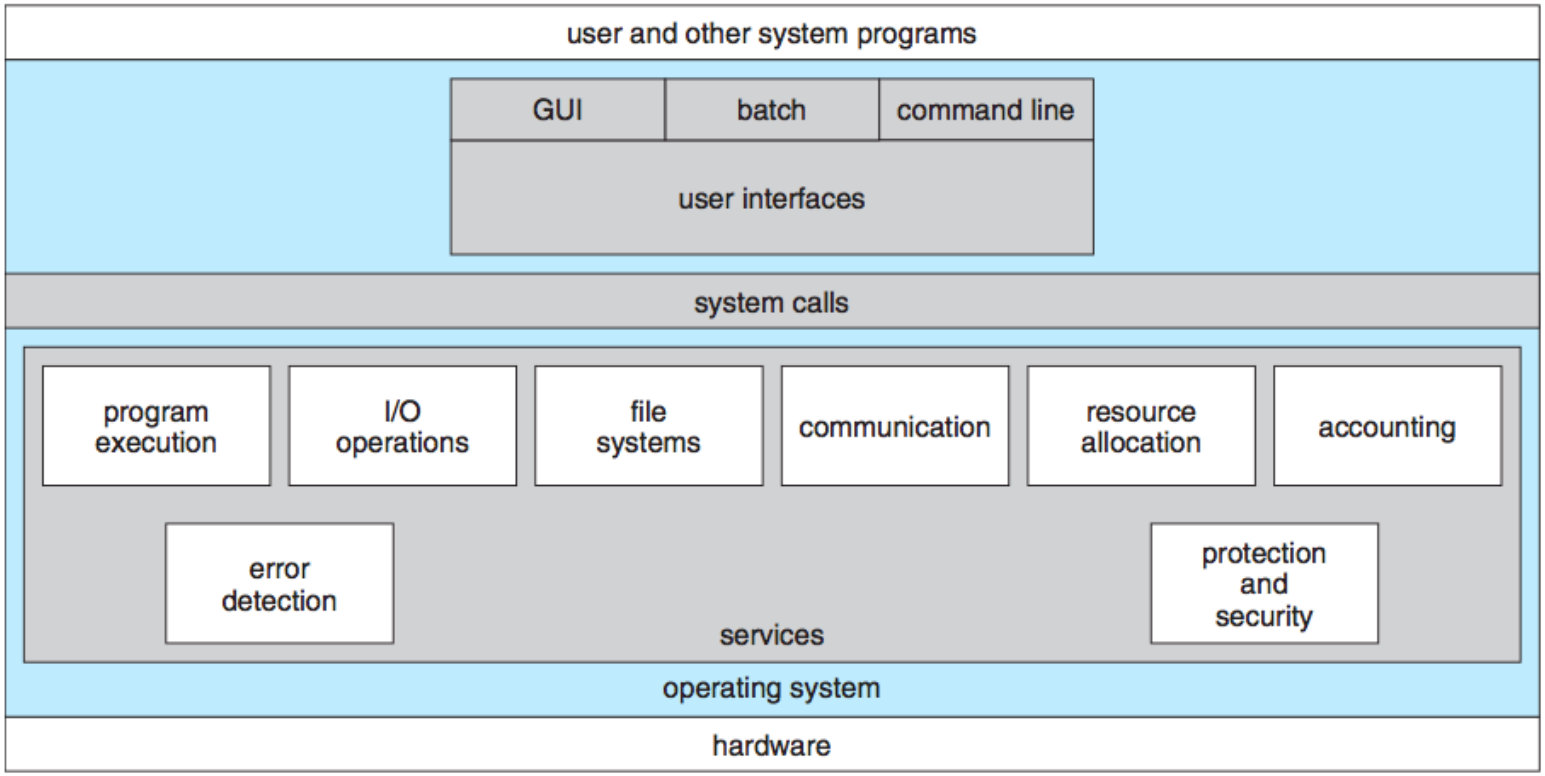
Faculty of Computer Science & Engineering
HCMC University of Technology

E-mail: hoai@hcmut.edu.vn
(partly based on slides of Le Thanh Van)

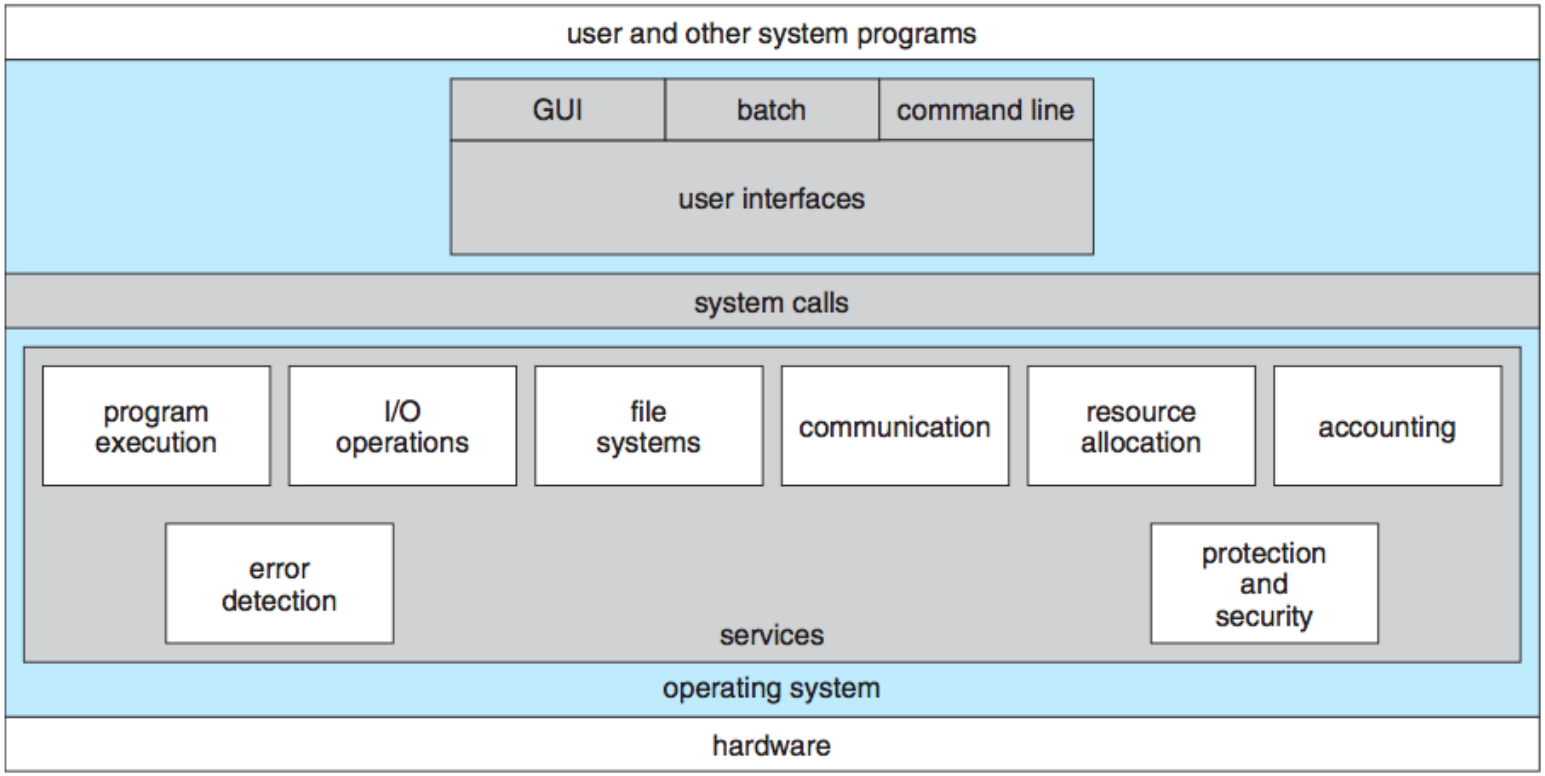
Outline

Outline

Operating system services

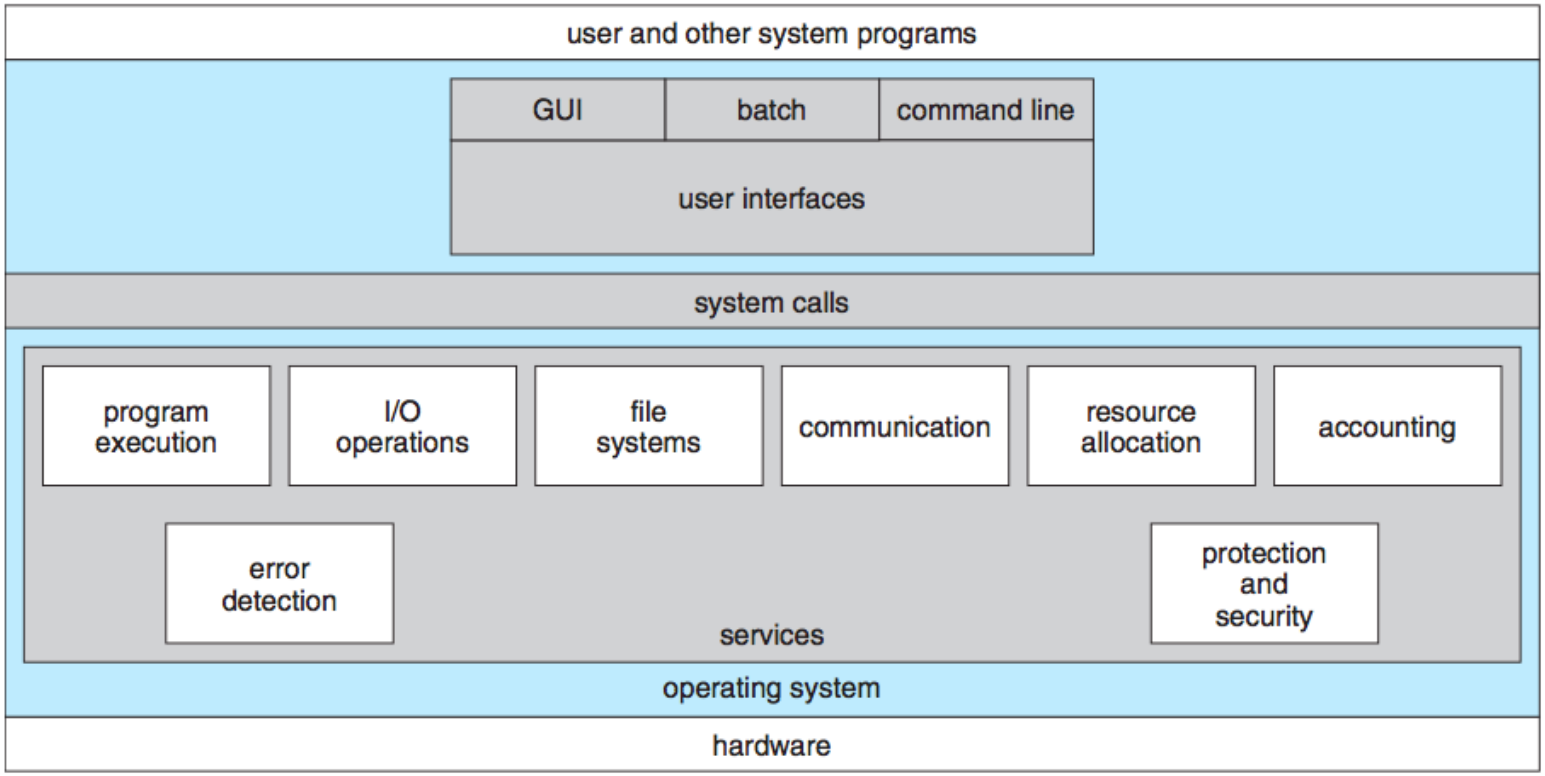


Operating system services



- Functional services
- Non-functional services

Operating system services



- Functional services
  - User interface (Graphical User Interface, Batch Interface, Command Line Interface), Program execution, I/O operations, File-system manipulation, Communications, Error detection
- Non-functional services
  - Resource allocation, Accounting, Protection and Security

What is a program ?

A computer program is a collection of instructions that performs a specific task when executed by a computer.

What is a program ?

A computer program is a collection of instructions that performs a specific task when executed by a computer.

- OS is able to load a program into memory and to run that program
- The program is able to end its execution (either normally or abnormally)

I/O operations

- A running program may access I/O devices, e.g., recording DVD)
- For efficiency and protection, OS must provide a means (for programs) to do I/O

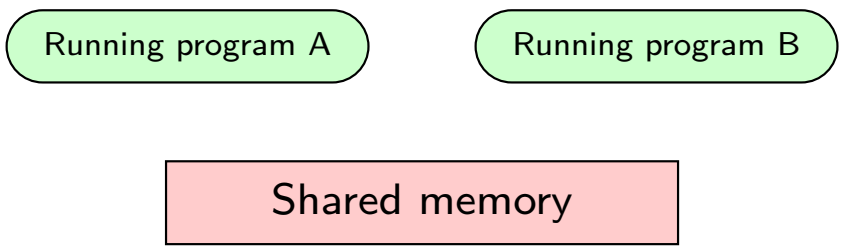
I/O operations

- A running program may access I/O devices, e.g., recording DVD)
- For efficiency and protection, OS must provide a means (for programs) to do I/O

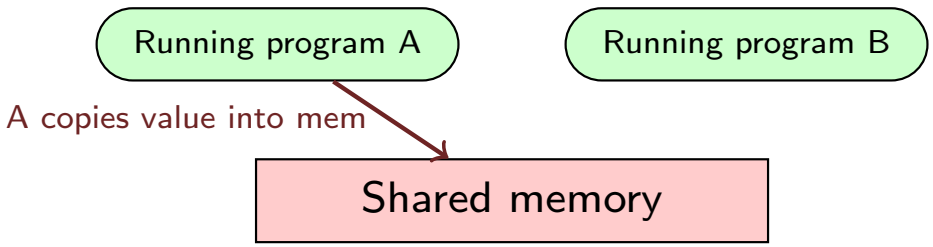
File-system manipulation

- Programs need operations on files/directories: list, create, delete, read, write, permission management

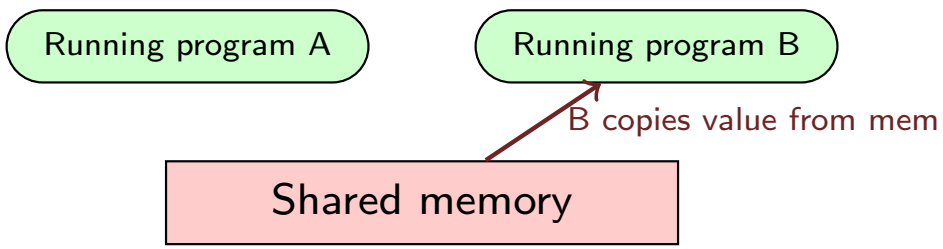
- There are many communications
  - between running programs on a machine
  - between running program on different machines
- There are two main types of communications
  - shared memory: read/write on a shared of memory
  - message passing: packets with predefined formats are exchanged between running programs



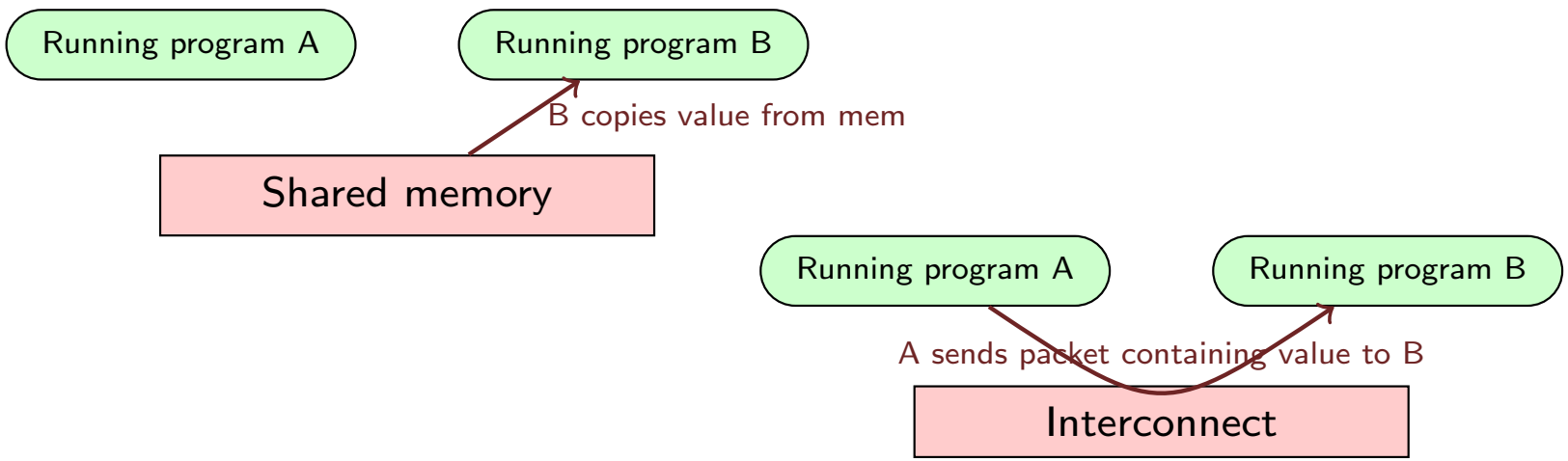
- There are many communications
  - between running programs on a machine
  - between running program on different machines
- There are two main types of communications
  - shared memory: read/write on a shared of memory
  - message passing: packets with predefined formats are exchanged between running programs



- There are many communications
  - between running programs on a machine
  - between running program on different machines
- There are two main types of communications
  - shared memory: read/write on a shared of memory
  - message passing: packets with predefined formats are exchanged between running programs



- There are many communications
  - between running programs on a machine
  - between running program on different machines
- There are two main types of communications
  - shared memory: read/write on a shared of memory
  - message passing: packets with predefined formats are exchanged between running programs



- Errors can occur everywhere
  - Hardware: CPU, memory, I/O devices,...
  - User program: arithmetic overflow, illegal access of memory, division by zero,...
- OS should take appropriate actions to detect and correct errors constantly.

Non-functional services

Resource allocation

- OS manages multiple resources (hardware, software) and allocates them to multiple users and multiple running programs
- Special codes are needed to make allocation efficiently, e.g., CPU scheduling, printers allocation

Accounting

- Recording which users to use how much and what kind of resources
- Usage statistics is useful to reconfigure for improvement of computing services

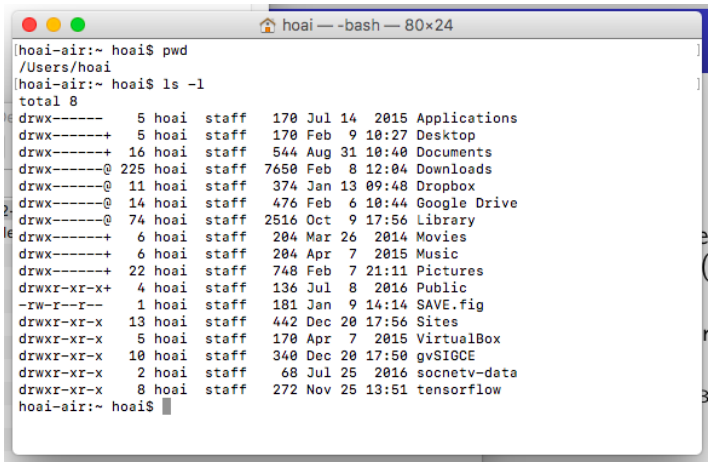
Protection and security

- Information of multiple users on a networked computer should be controlled by its owner.

User interface

- Command interpreter - a special program to allows users to directly enter commands (or programs) (by text) to be performed by the operating system
  - Multiple command interpreters (also shell) in a modern operating system

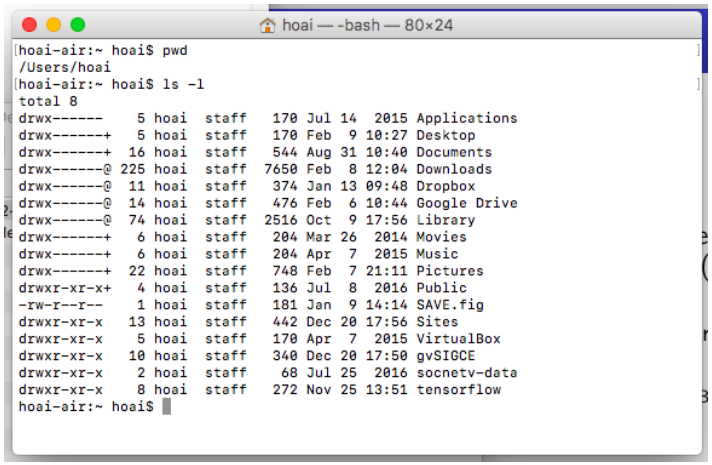
    Example: Bourne shell, Bash shell, C shell, Bourne-Again shell, Korn shell



User interface

- Command interpreter - a special program to allows users to directly enter commands (or programs) (by text) to be performed by the operating system
  - Multiple command interpreters (also shell) in a modern operating system

    Example: Bourne shell, Bash shell, C shell, Bourne-Again shell, Korn shell



- GUI - a user friendly graphical interface, input/output is performed in a more interactive way

Choice of interface

Interface choice

Kind of interface (CLI or GUI) is mostly one of personal preference

- System administratiors or power users often prefer CLI
  - With deep system knowledge, using CLI is more efficient and secure
  - Multiple shell commands can be combined into a program, called shell script to perform more complex tasks.
- Normal users often choose GUI

Choice of interface

Interface choice

Kind of interface (CLI or GUI) is mostly one of personal preference

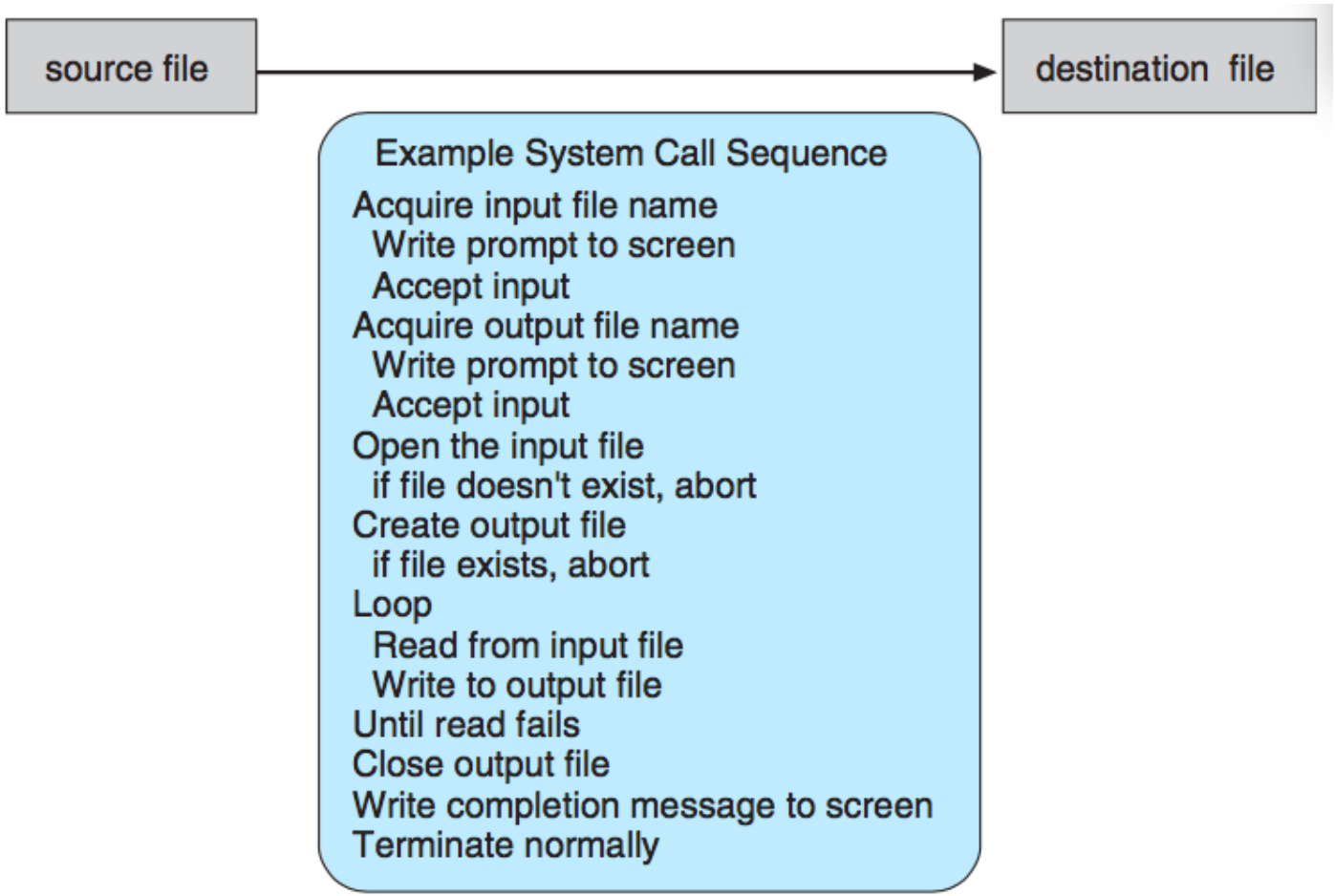
- System administratiors or power users often prefer CLI
  - With deep system knowledge, using CLI is more efficient and secure
  - Multiple shell commands can be combined into a program, called shell script to perform more complex tasks.
- Normal users often choose GUI

UI can vary from system to system. Therefore, it is not a direct function of operating system.

Outline

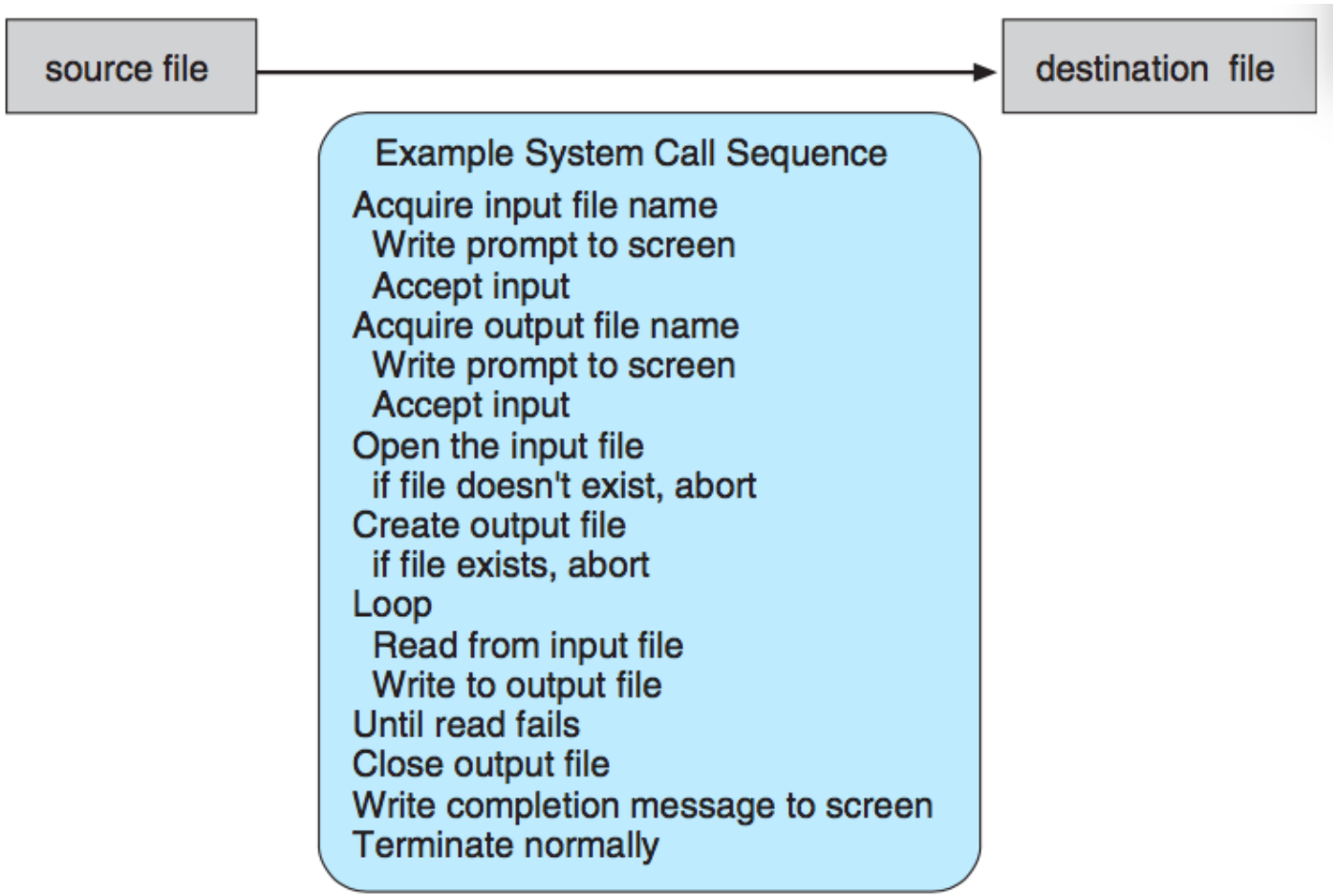
Example on system call demand

File to file copy



Example on system call demand

File to file copy



Even a simple program executes many system calls

API vs. system call interface

Application Programming Interface (API)

Set of functions available to application programmers

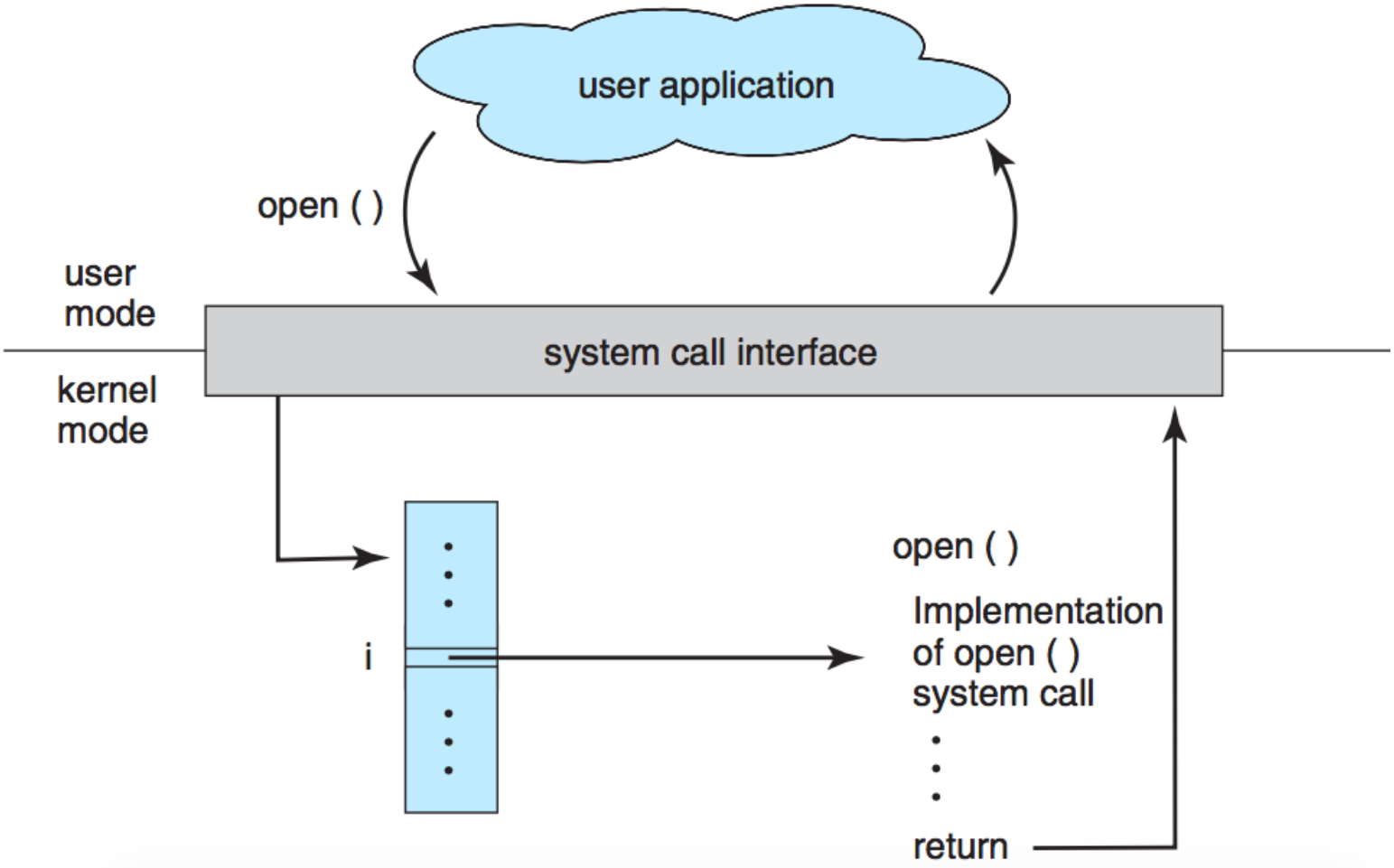
- Using API to increase program portability: ability to compile and run on systems with the same API without code modification
- Working with API is easier than with actual system calls
- 3 most common APIs: Windows API, POSIX API, Java API

System-call interface

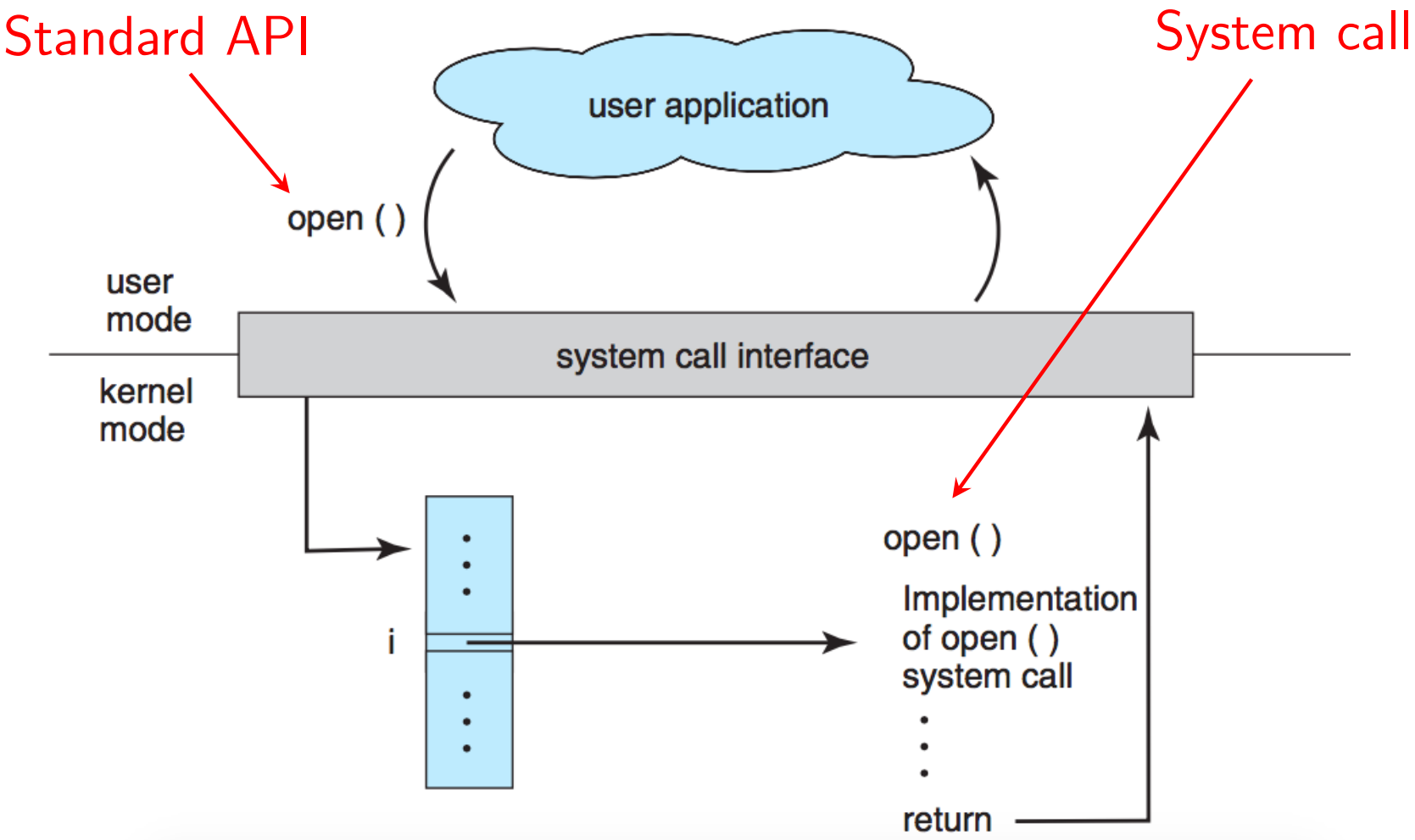
A run-time support system (e.g., `libc`) serves as a link between API and system calls in operating system

- Each system call is identified by an index number

A call to system call `open()`



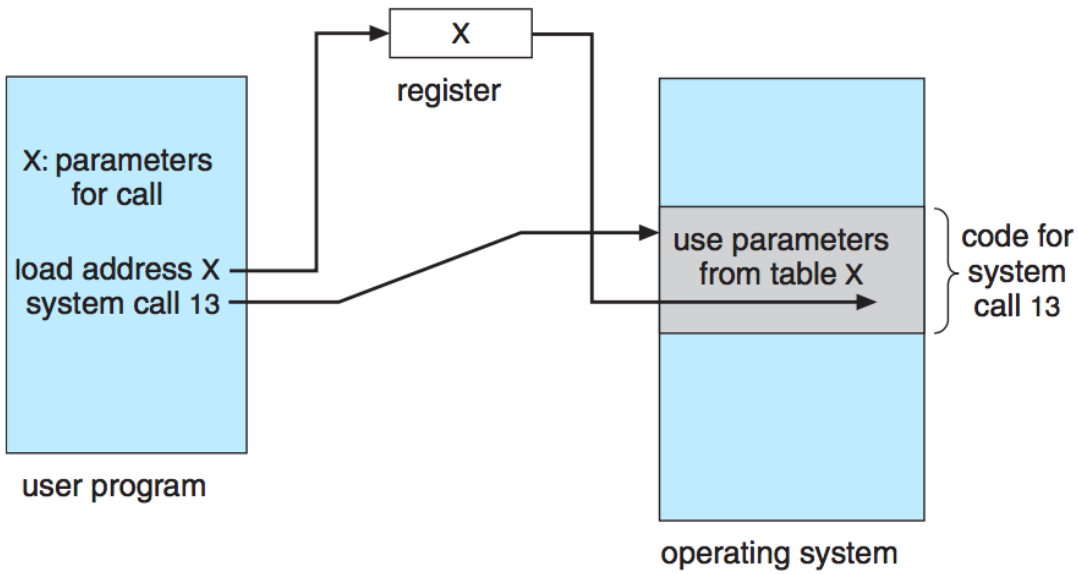
A call to system call `open()`



Parameters passing to system call

There are 3 ways.

- By registers: not enough for large parameters
- Stored in block (or table), passing address of the block in parameter



- Pushed onto program's stack, and popped off by OS

Types of system calls

- Process control
- File management
- Device management
- Information maintenance
- Communications
- Protection

Process control

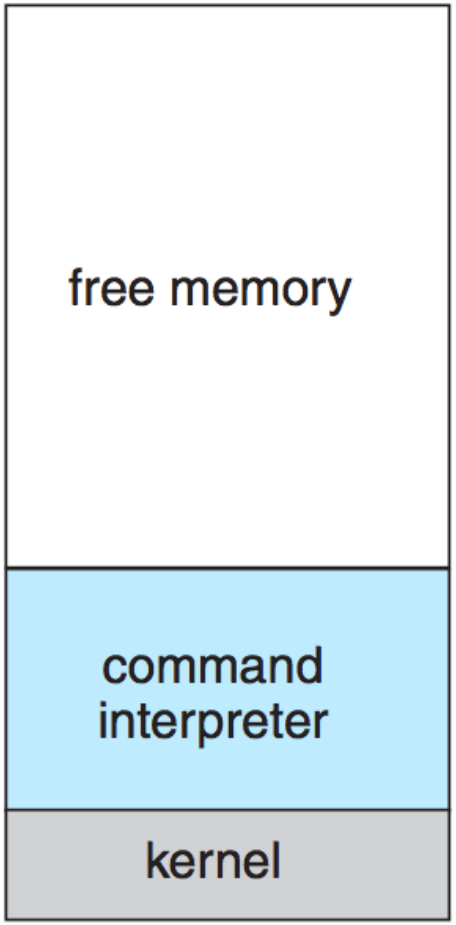
What is a process ?

A program loaded into memory and executing is called a process

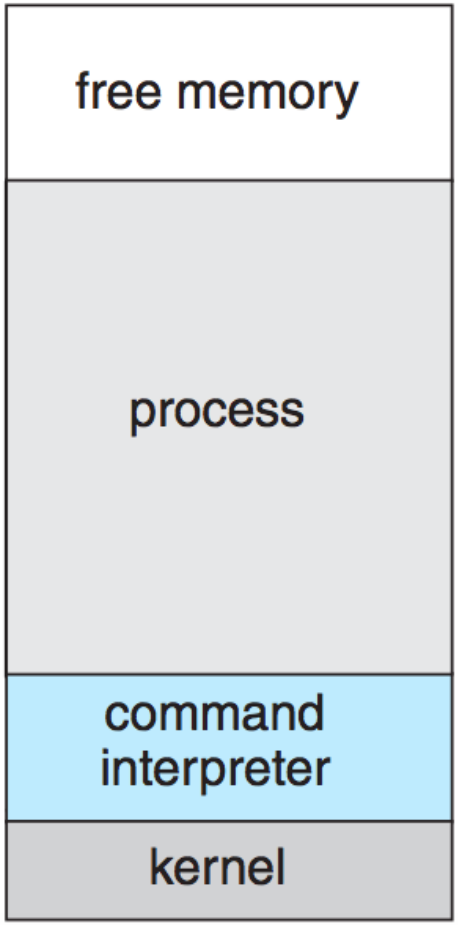
- `end()`, `abort()`
- `load()`, `execute()`
- `create_process()`, `terminate_process()`
- `get_process_attributes()`, `set_process_attributes()`
- `wait_for_time()`
- `wait_event()`, `signal_event()`
- allocate and free memory

Execution on single-task OS

MS DOS



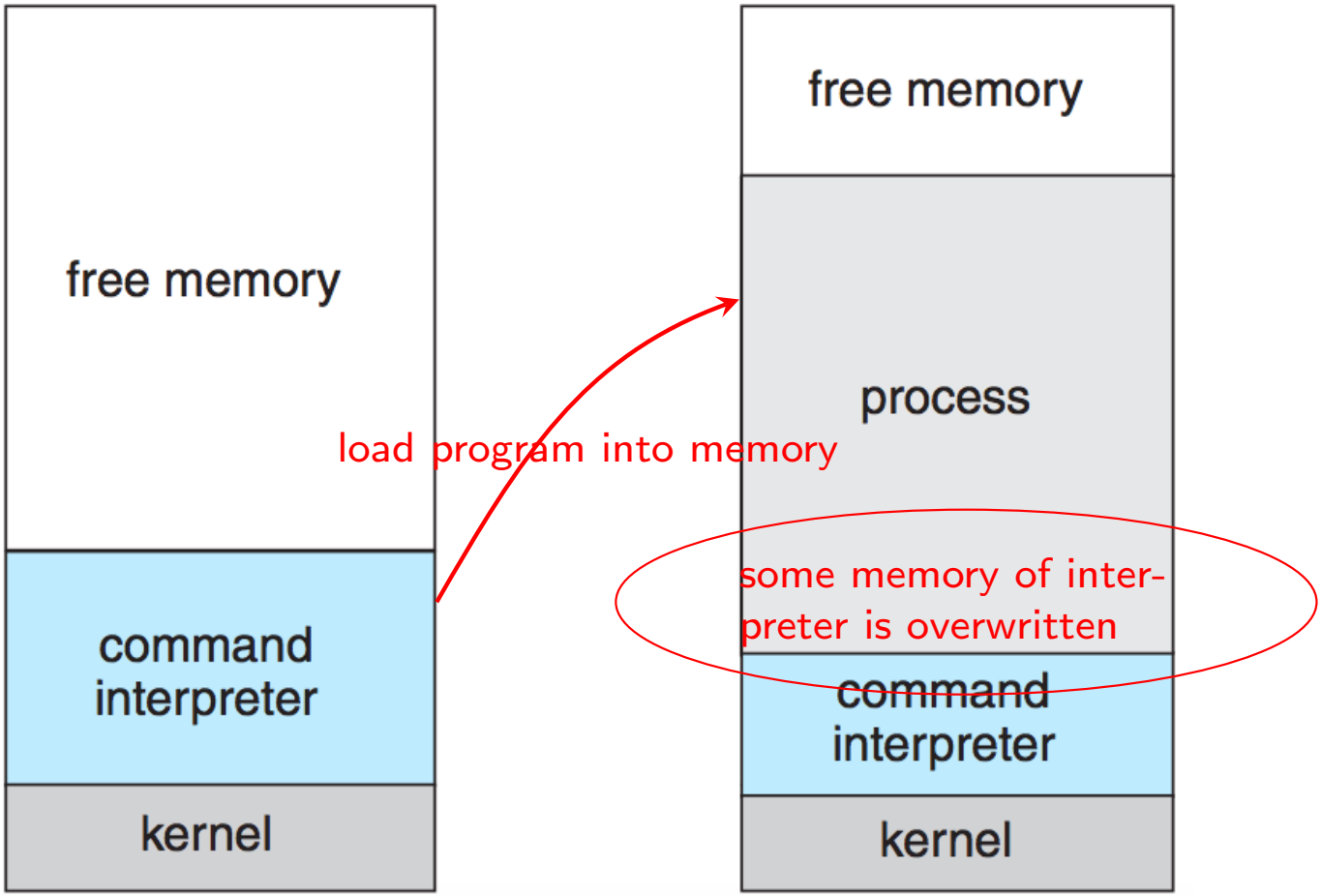
At system startup



Running a program

Execution on single-task OS

MS DOS

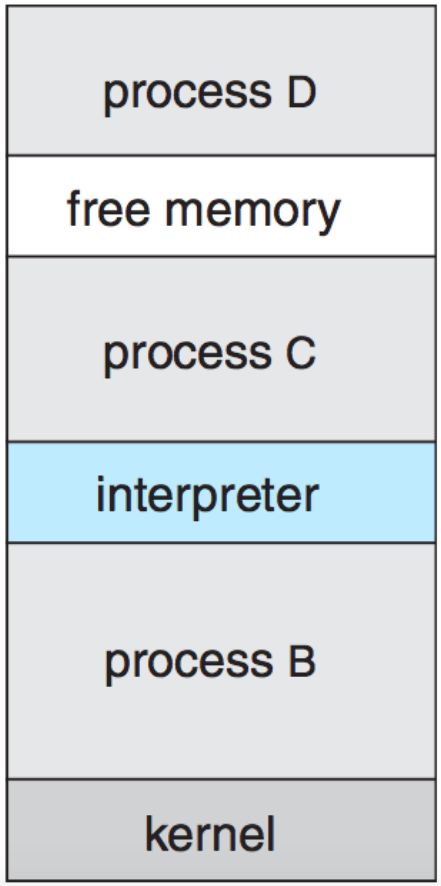


At system startup

Running a program

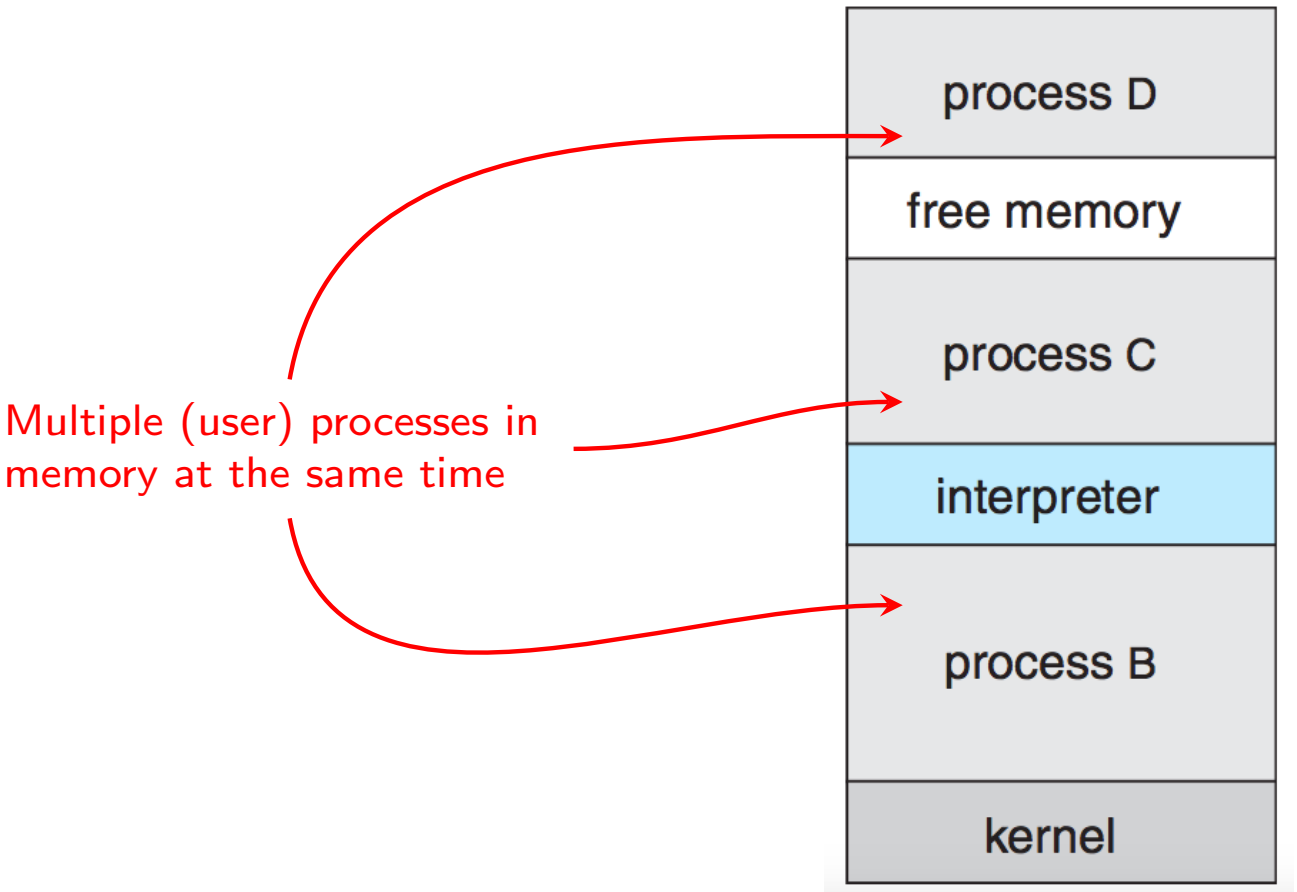
Execution on multiple-task OS

FreeBSD



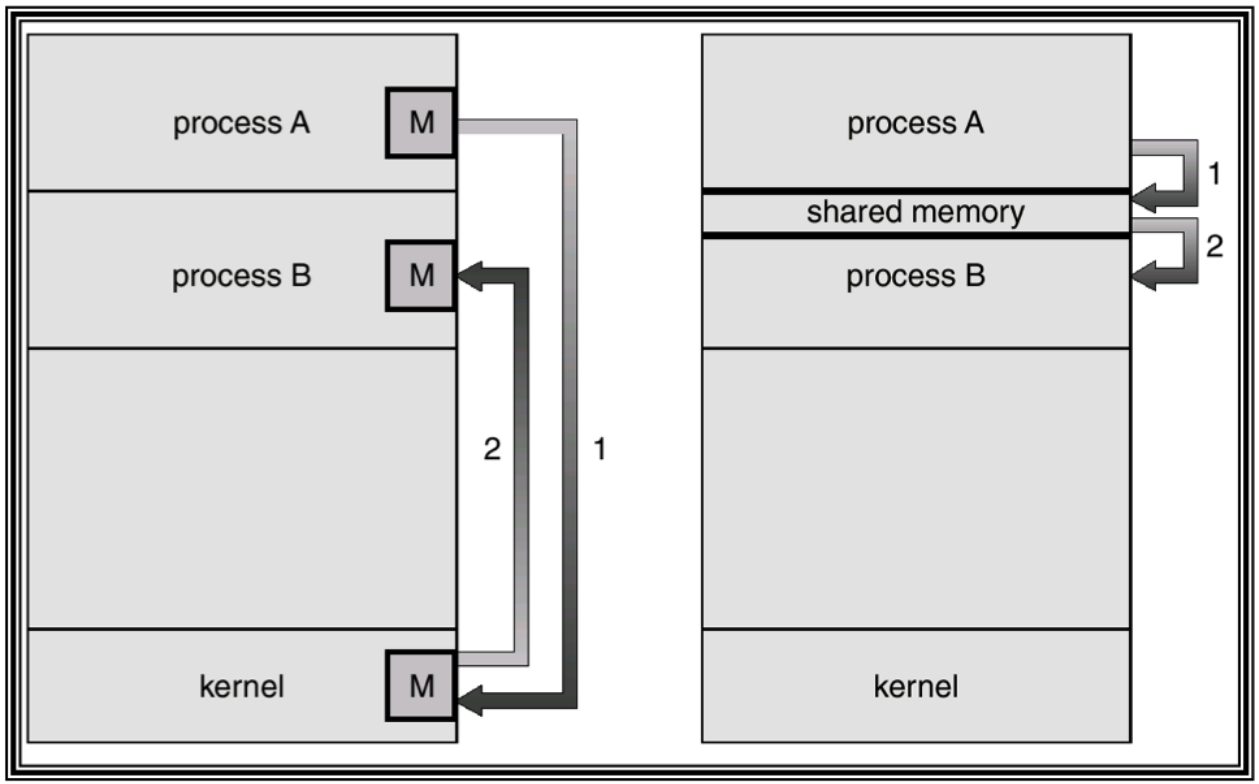
Execution on multiple-task OS

FreeBSD



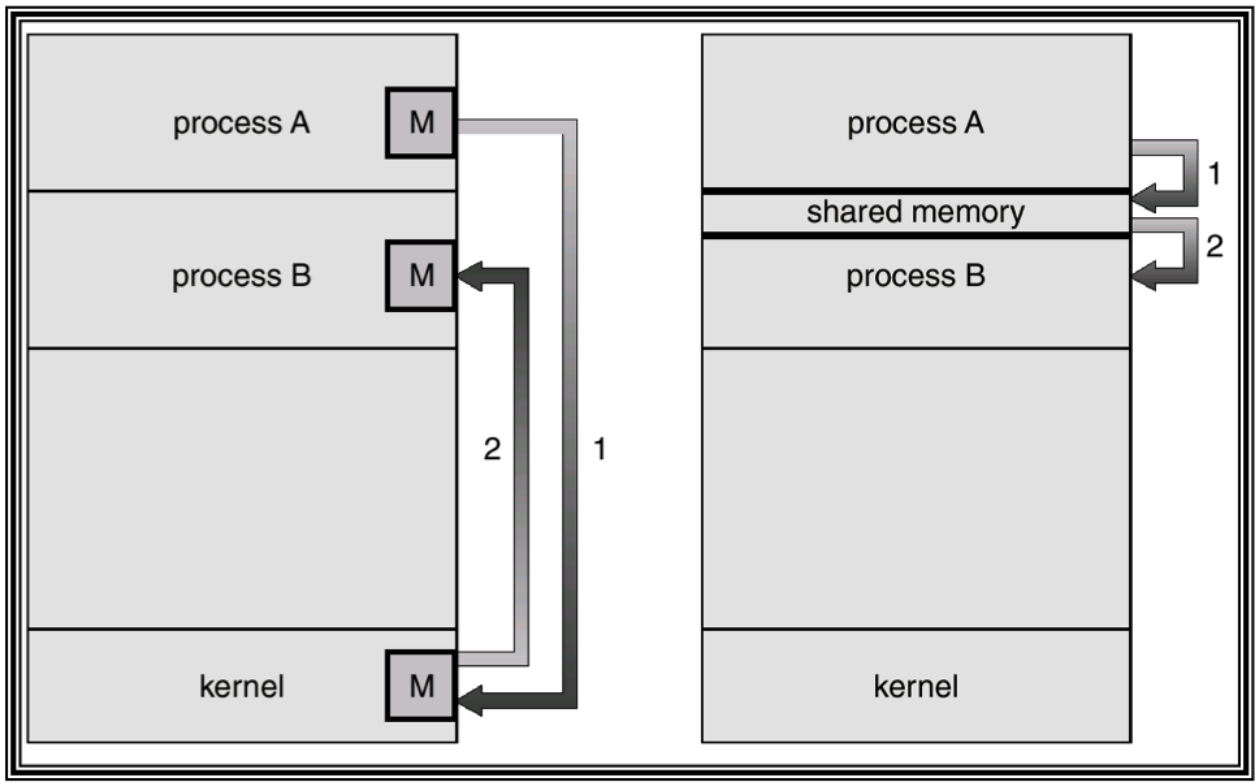
Interprocess communication

- Interprocess communication may take place using either message passing or shared memory



Interprocess communication

- Interprocess communication may take place using either message passing or shared memory



Message passing

Shared memory

System programs

System programs = system utilities

Providing a convenient environment for program development and execution

There are categories as follows.

- File manipulation
- Status information
- File modification
- Programming-language support
- Program loading and execution
- Communications
- Background services

System programs

System programs = system utilities

Providing a convenient environment for program development and execution

There are categories as follows.

- File manipulation
- Status information
- File modification
- Programming-language support
- Program loading and execution
- Communications
- Background services

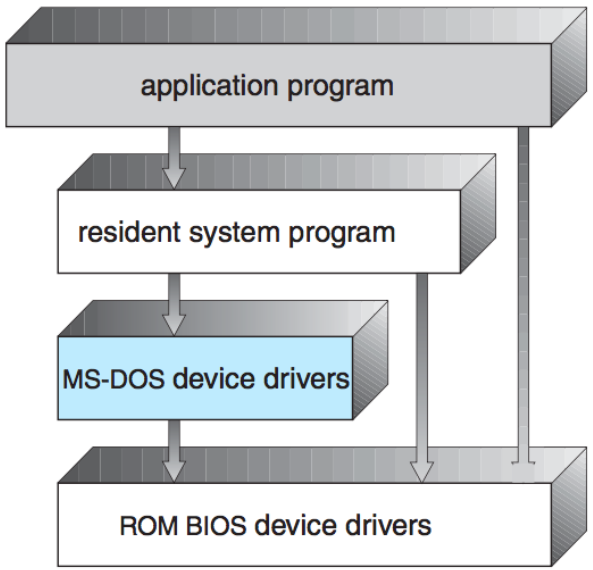
The view of most users on the operating system is defined by system programs, not actual system calls.

Outline

Simple structure

MS-DOS

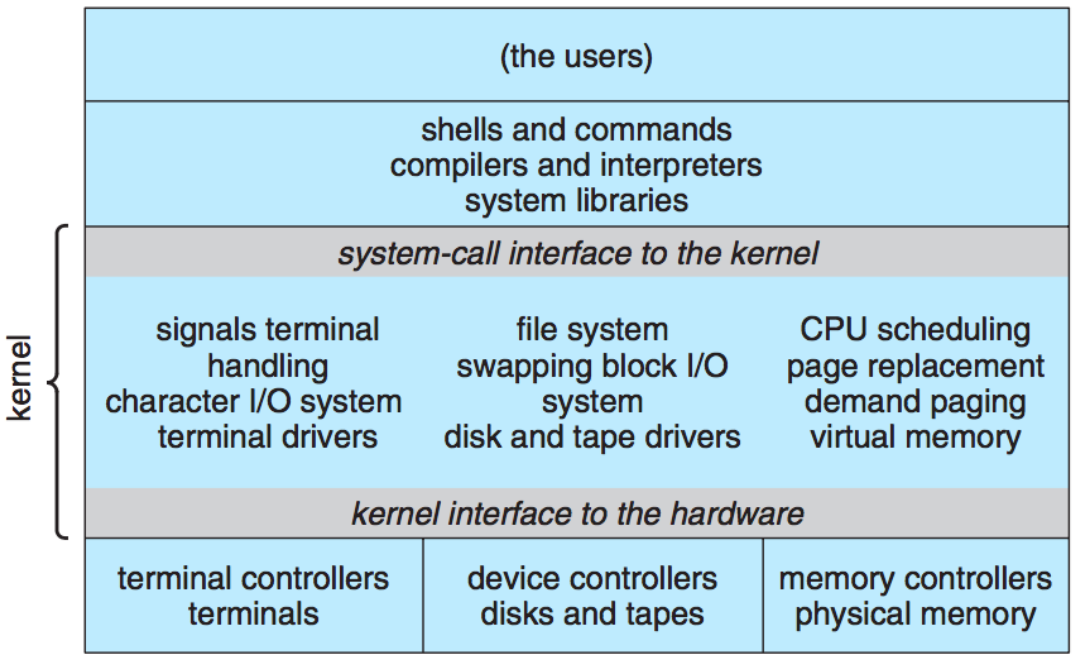
- MS-DOS: written to provide the most functionality in the least space
  - not divided into modules
  - interfaces and levels of functionality are not well separated



Simple structure

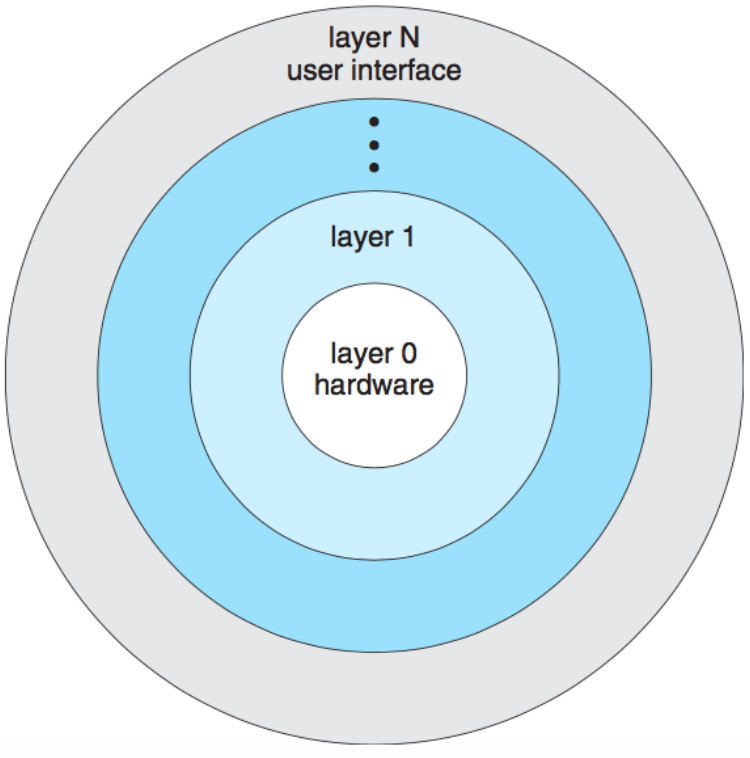
Traditional UNIX

- Limited structure with limited functionality
- Consists of 2 parts: kernel and system programs



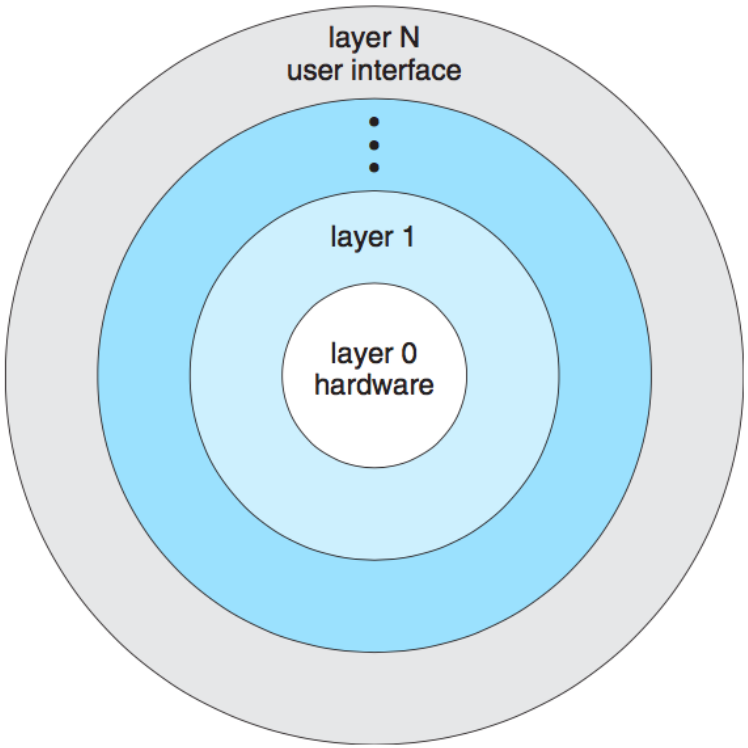
- Considered to be layered to some extent: system-call inteface, kernel and hardware interface
- Enormous amount of functionality combined in one level → monolithic structure

Layered approach



- OS is broken into several layers
- Layer *M* provides data structures & routines for upper layer, and can invoke operations of lower layer

Layered approach



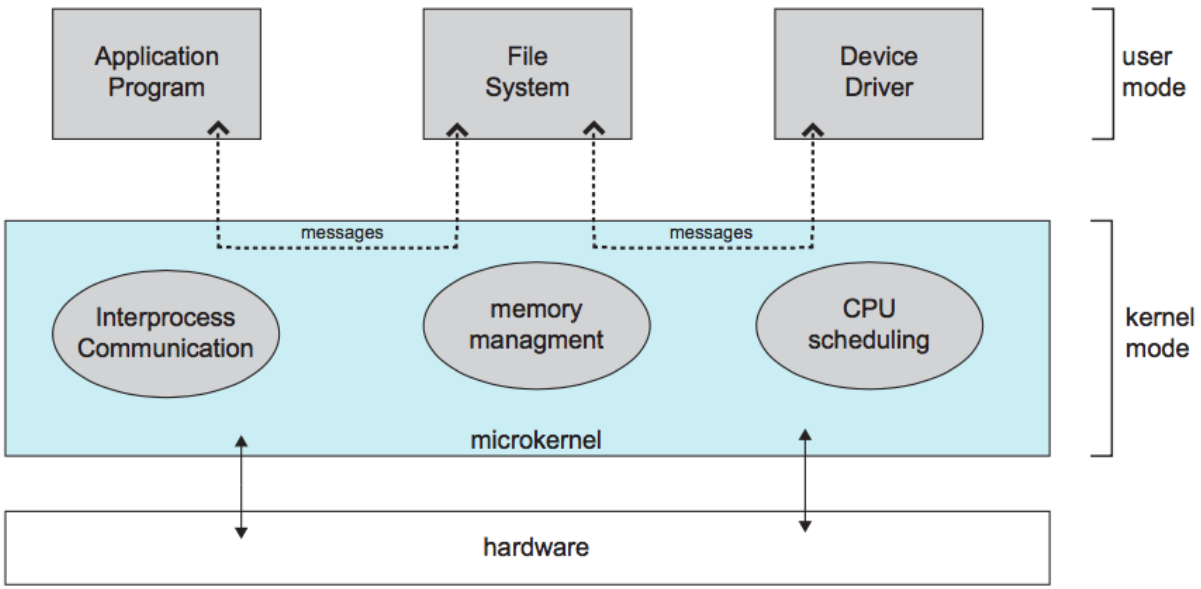
- OS is broken into several layers
- Layer M provides data structures & routines for upper layer, and can invoke operations of lower layer

Advantages (due to modularity): easy to debug; simple in design and implementation; secure

Disadvantages: Not easy to define layers (which layer is above/below which layer); inefficiency

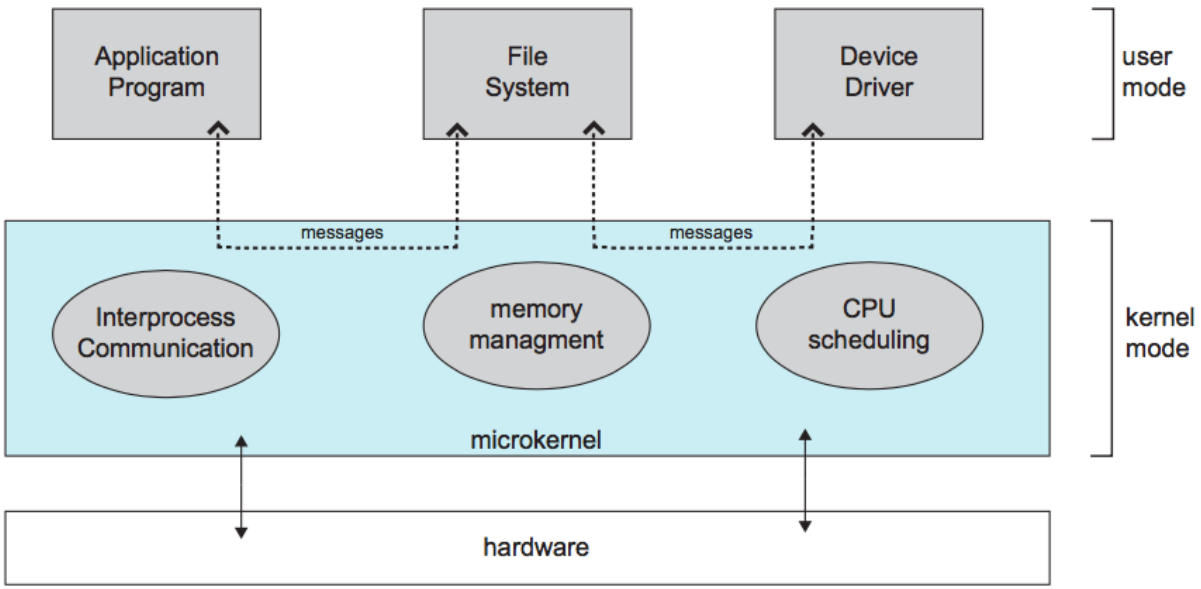
Microkernel structure

- Moves as much from the kernel into "user" space
- Communication takes place between user modules using message passing



Microkernel structure

- Moves as much from the kernel into "user" space
- Communication takes place between user modules using message passing

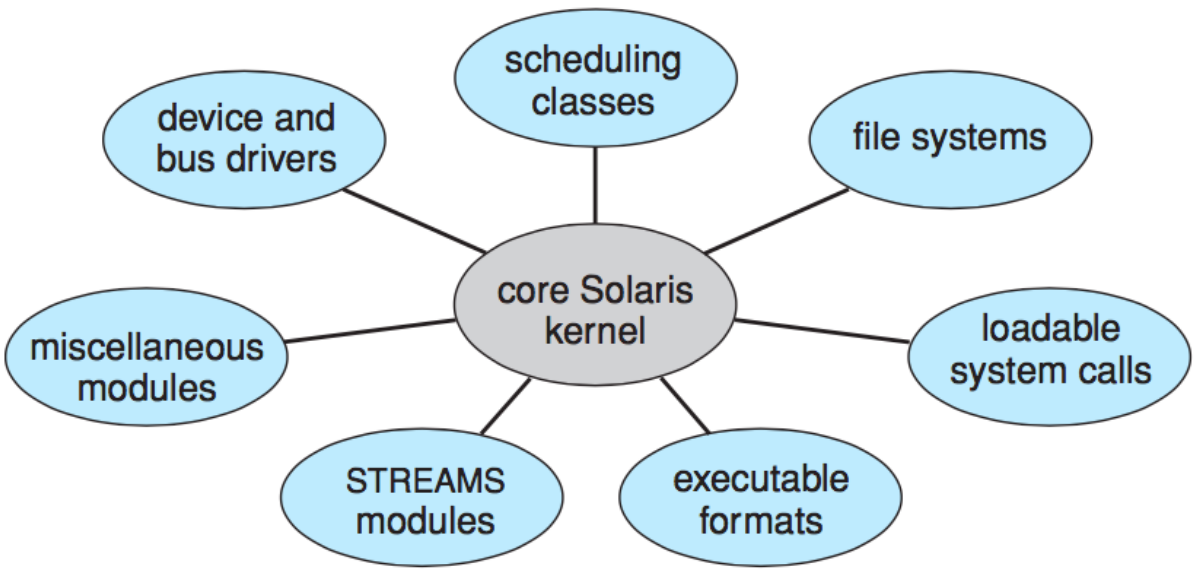


Advantages (due to modularity): easy to extend kernel; easier to port OS to new architectures; more reliable (less code running in kernel); more secure

Disadvantages: system-function overhead

Modules-based structure

- Best current methodology is loadable kernel modules
  - Only need core services
  - Additional services can be loaded as modules in boot time and run time
  - Additional services have to be recompiled to add new features
- Idea of modules-based kernel more flexible than layered approach and also similar to microkernel structure



Hybrid structure

- In practice, very few OS adopt a single, stricted defined structure

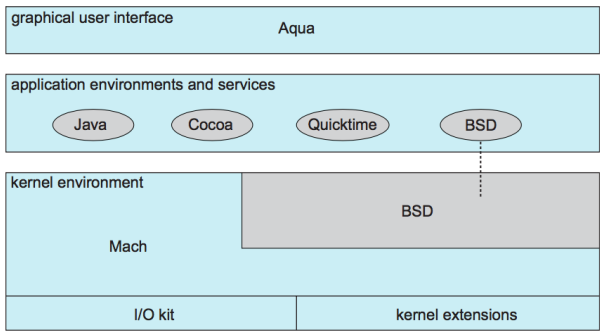


Figure 2.16 The Mac OS X structure.

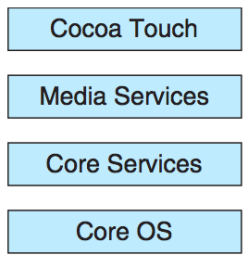


Figure 2.17 Architecture of Apple's iOS.

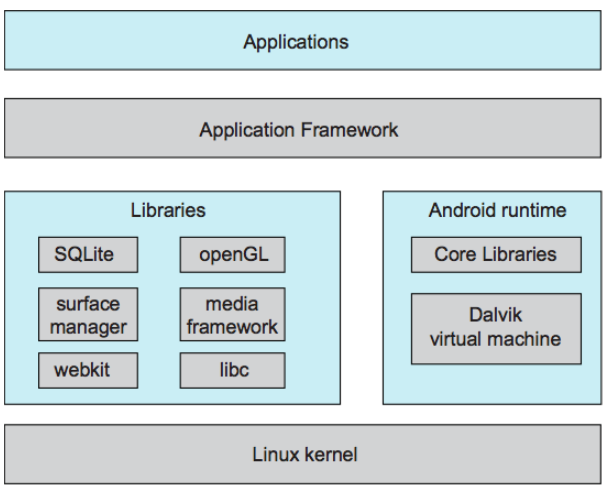


Figure 2.18 Architecture of Google's Android.

Outline