Chapter 1: Operating-System Structures

Chapter 1: Operating-System Structures

- Operating System Services
- System Calls
- Types of System Calls
- Operating System Structure
- System Boot

Operating System Services

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

- OS Services provided to the *users* to make their task easier**(U, P, I, F, C,E)**
  - **User interface** - Almost all operating systems have a user interface (**UI**).
    - ***Command-Line (CLI): a pgm that allows text commands.***
    - ***Graphics User Interface (GUI): use of menus, selections,pointing device to direct I/O.***
    - ***Batch interface: commands are entered into files, files are executed***
  - **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. Users do not directly control I/O, OS must do it.

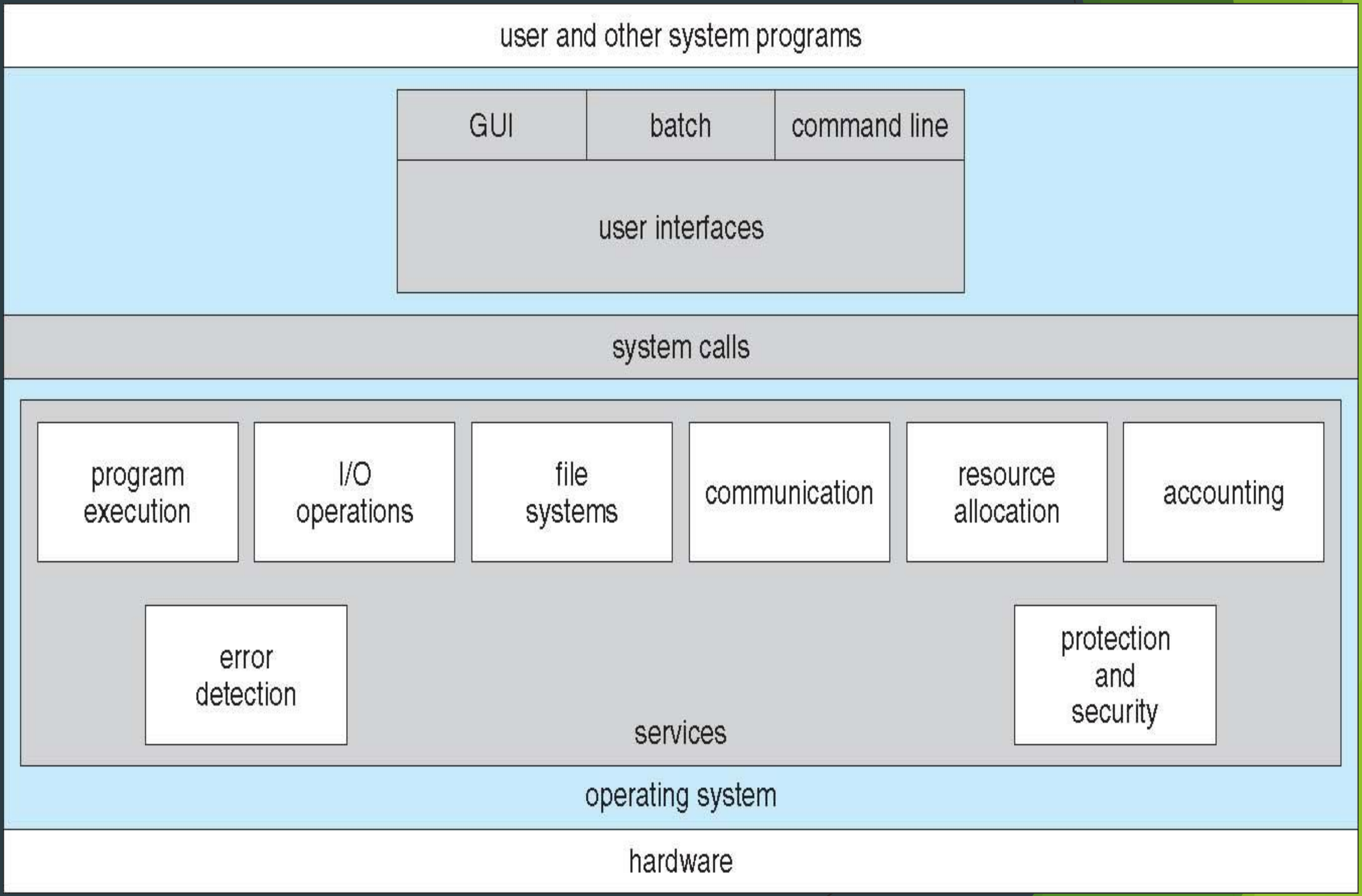
Operating System Services (Cont.)

- One set of operating-system services provides functions that are helpful to the user (Cont.):
  - **File-system manipulation** - Programs need to read and write files and directories, create and delete them, search them, list file Information, and 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).
  - **Error detection** – OS needs to be constantly aware of possible errors
    - May occur in the CPU and memory hardware(memory error/power failure); I/O devices(conn failure in n/w, lack of paper in printer); user program(arithmetic overflow, access illegal memory location, too great use of CPU).
    - For each type of error, the OS should take the appropriate action to ensure correct and consistent computing
    - Debugging facilities can enhance -the user's and programmer's abilities to efficiently use the system.

Operating System Services (Cont.)

- OS functions *to ensure the efficient operation of the system itself*
  - **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
  - **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, and extends to defending external I/O devices from invalid access attempts Resource-sharing

A View of Operating System Services



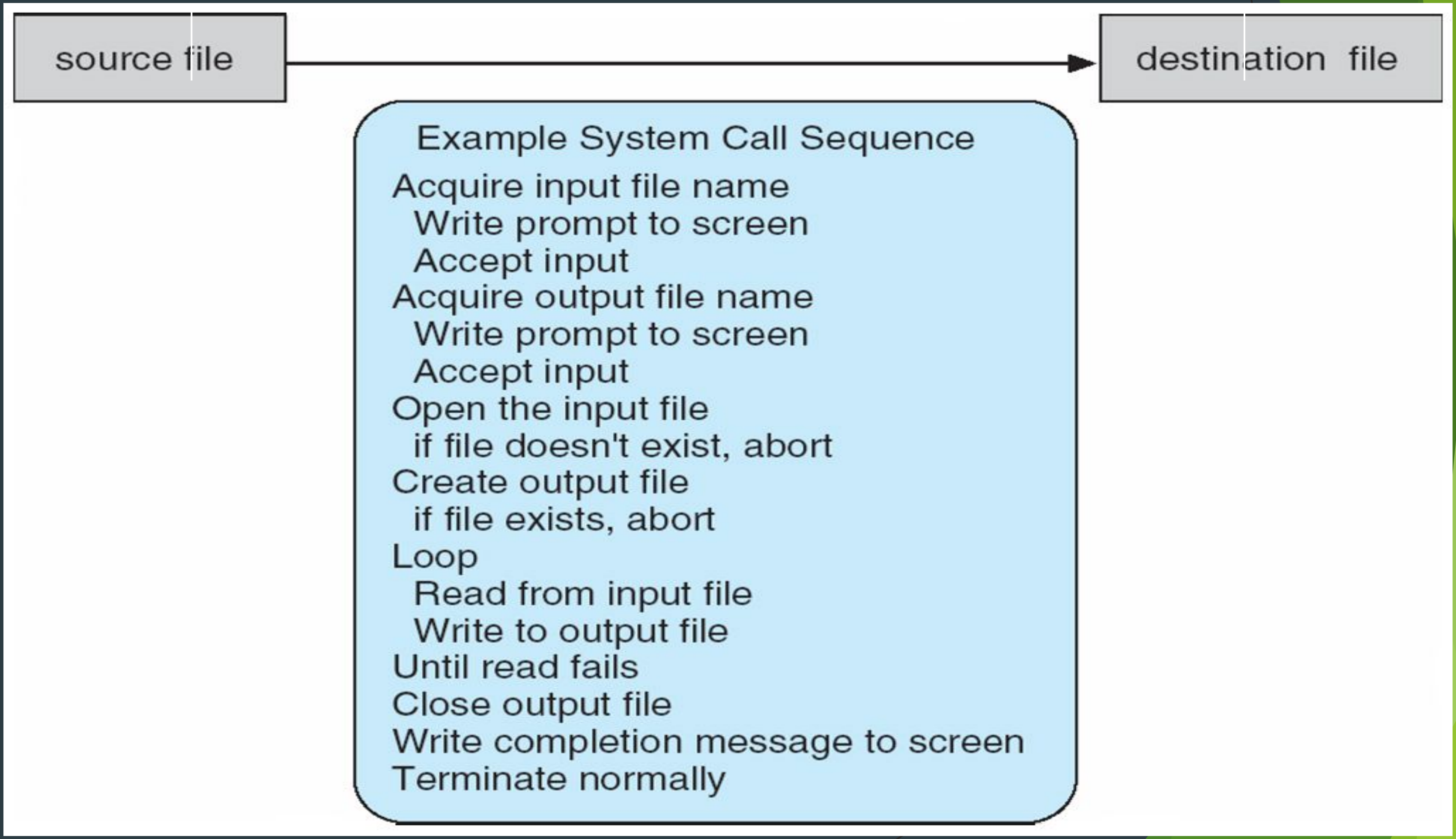
System Calls

- 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.
- The 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),
- Java API for the Java virtual machine (JVM)

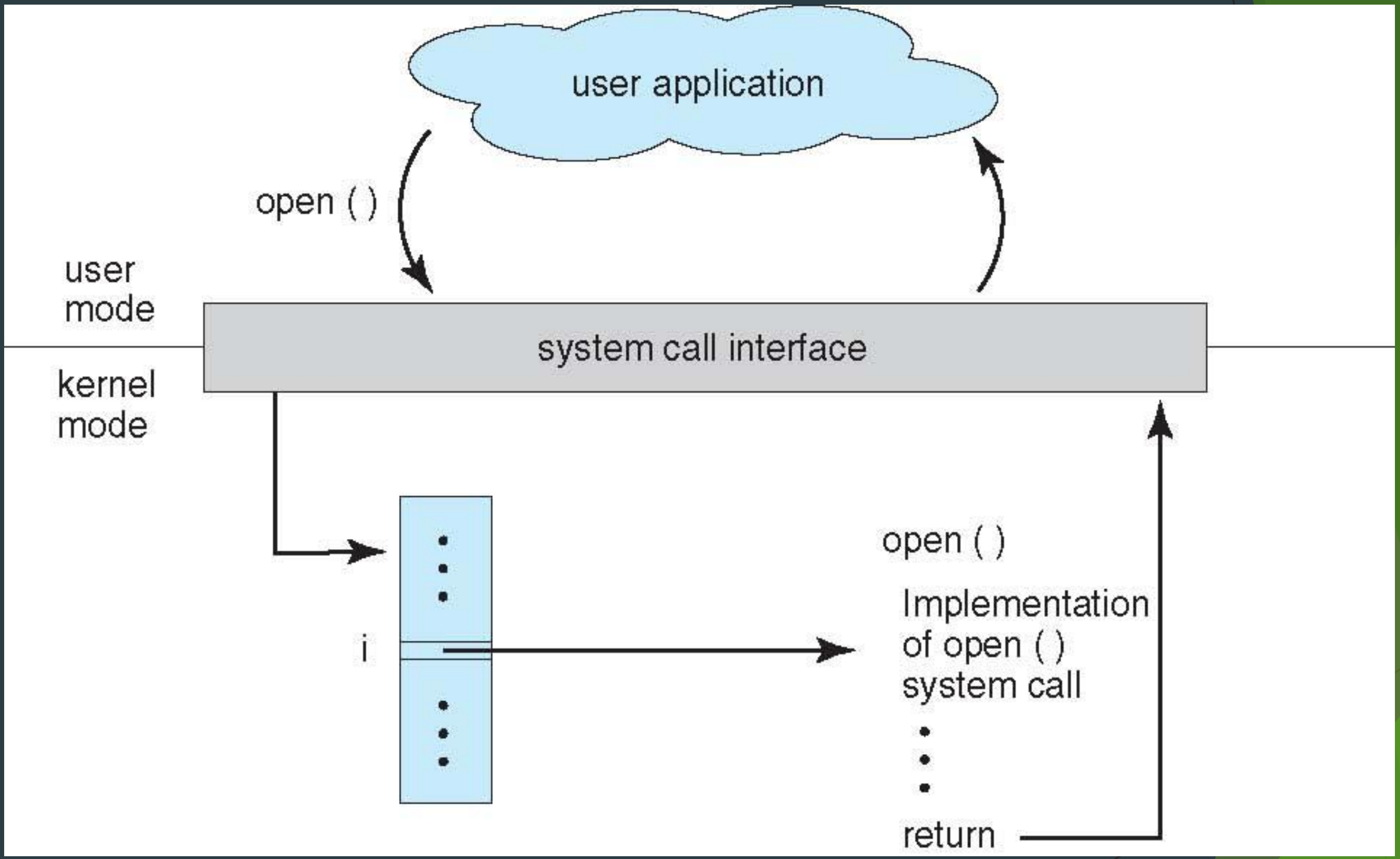
Note that the system-call names used throughout this text are generic

Example of System Calls

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



API – System Call – OS Relationship



Types of System Calls

- 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
  - Dump memory if error
  - Debugger for determining bugs, single step execution
  - Locks for managing access to shared data between processes

Types of System Calls

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

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

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, and receive messages if the **message passing model** to the **hostname** 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 (Cont.)

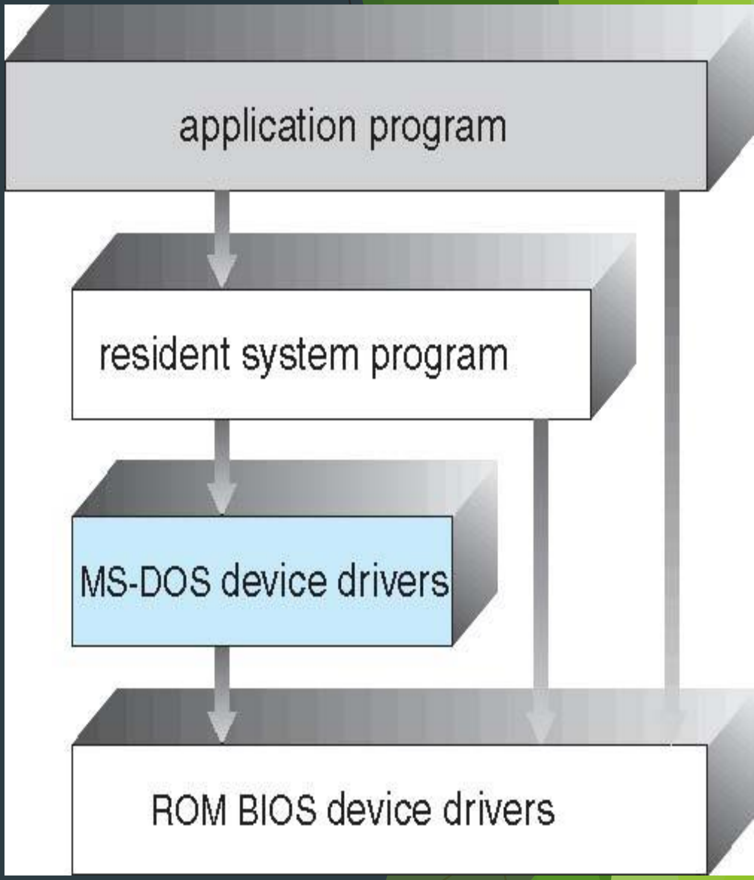
- Protection
  - Control access to resources
  - Get and set permissions
  - Allow and deny user access

Operating System Structure

- General-purpose OS is a very large program
- Various ways to structure one
  - Simple structure – MS-DOS
  - Layered – an abstraction

Simple Structure -- MS-DOS

- MS-DOS – written to provide the most functionality in the least space
  - Not divided into modules
  - Although MS-DOS has some structure, its interfaces and levels of functionality are not well separated
  - It has a monolithic structure.
  - The user programs can directly access the basic I/O routines
  - Hence, if user pgm fails, the system crashes.



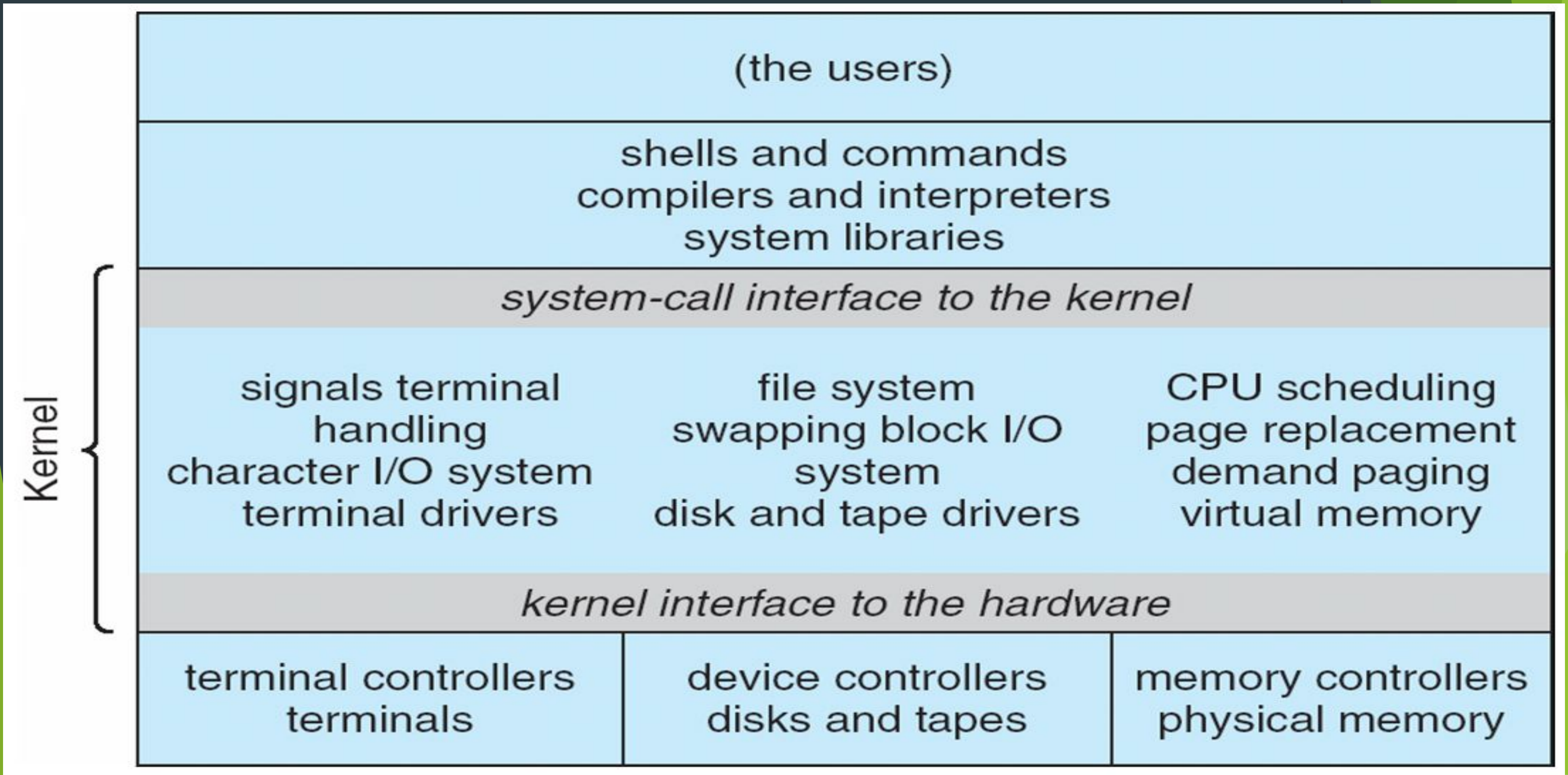
Non Simple Structure -- UNIX

UNIX – limited by hardware functionality, the original UNIX operating system had limited structuring. 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

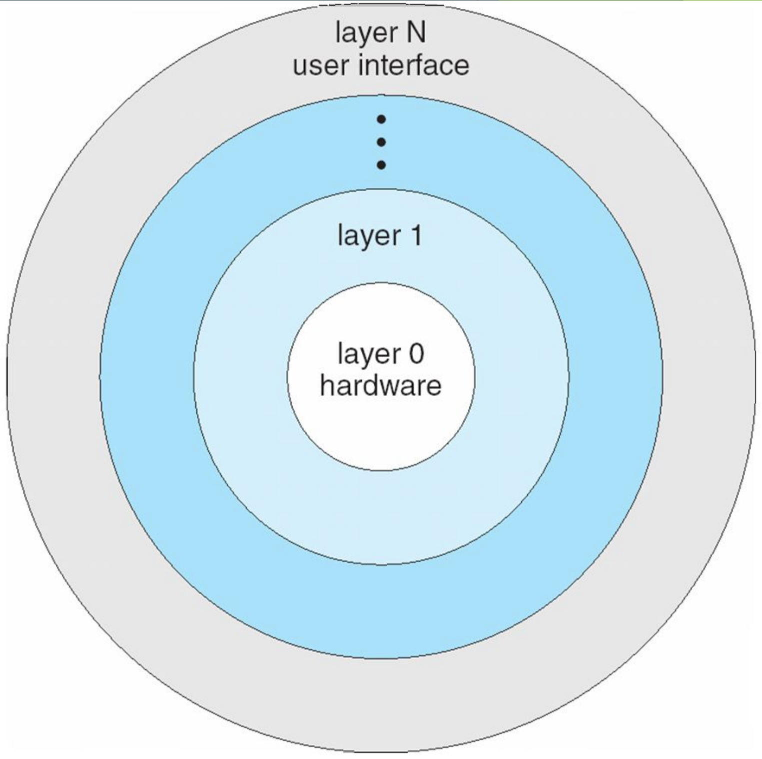
Traditional UNIX System Structure

- Traditional UNIX OS –limited by h/w functionality.
- Consists of 2 parts: the kernel and the system pgms.
- Kernel consists of device drivers and interfaces.
- The layer *below system call interface* and *above physical h/w* is the kernel .
- The kernel functionality is enormous ;difficukt to implement and maintain



Layered Approach

- 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



System Boot

- When power is initialized on the system, execution starts at a fixed memory location
  - Firmware ROM used to hold the initial boot code
- Operating system must be made available to hardware so hardware can start it
  - Small piece of code – **bootstrap loader**, stored in **ROM** or **EEPROM** locates the kernel, loads it into memory, and starts it
  - Sometimes two-step process where the **boot block** at a fixed location is loaded by ROM code, which loads the bootstrap loader from the disk
- Common bootstrap loader, **GRUB**, allows selection of kernel from multiple disks, versions, kernel options
- Kernel loads and the system is then **running**

To be explored:

RAM
ROM
Primary Memory
Cache
Secondary Memory
Firmware
Device Drivers
Disadvantages of the layered approach of OS
Distributed systems
I/O routines