

CSE-309

Operating Systems

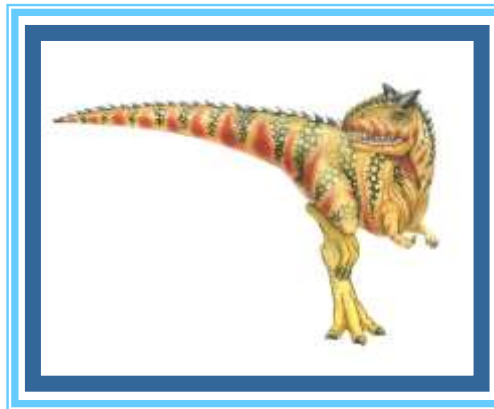
Mohammad Shariful Islam
Lecturer, Department of CSE

Mobile: 01747612143

Email: sharifulruhan@gmail.com

Chapter 2:

Operating-System Structures





Chapter 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 Debugging
- Operating System Generation
- System Boot





Objectives

- To describe the services an operating system provides to users, processes, and other systems
- To discuss the various ways of structuring an operating system
- To explain how operating systems are installed and customized and how they boot





Operating System Services

- Operating systems provide an environment for execution of programs and services to programs and users.
- 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).
 - ▶ Varies between **Command-Line (CLI)**, **Graphics User Interface (GUI)**, **Batch**, **web-based**.
 - **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 (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, 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, 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





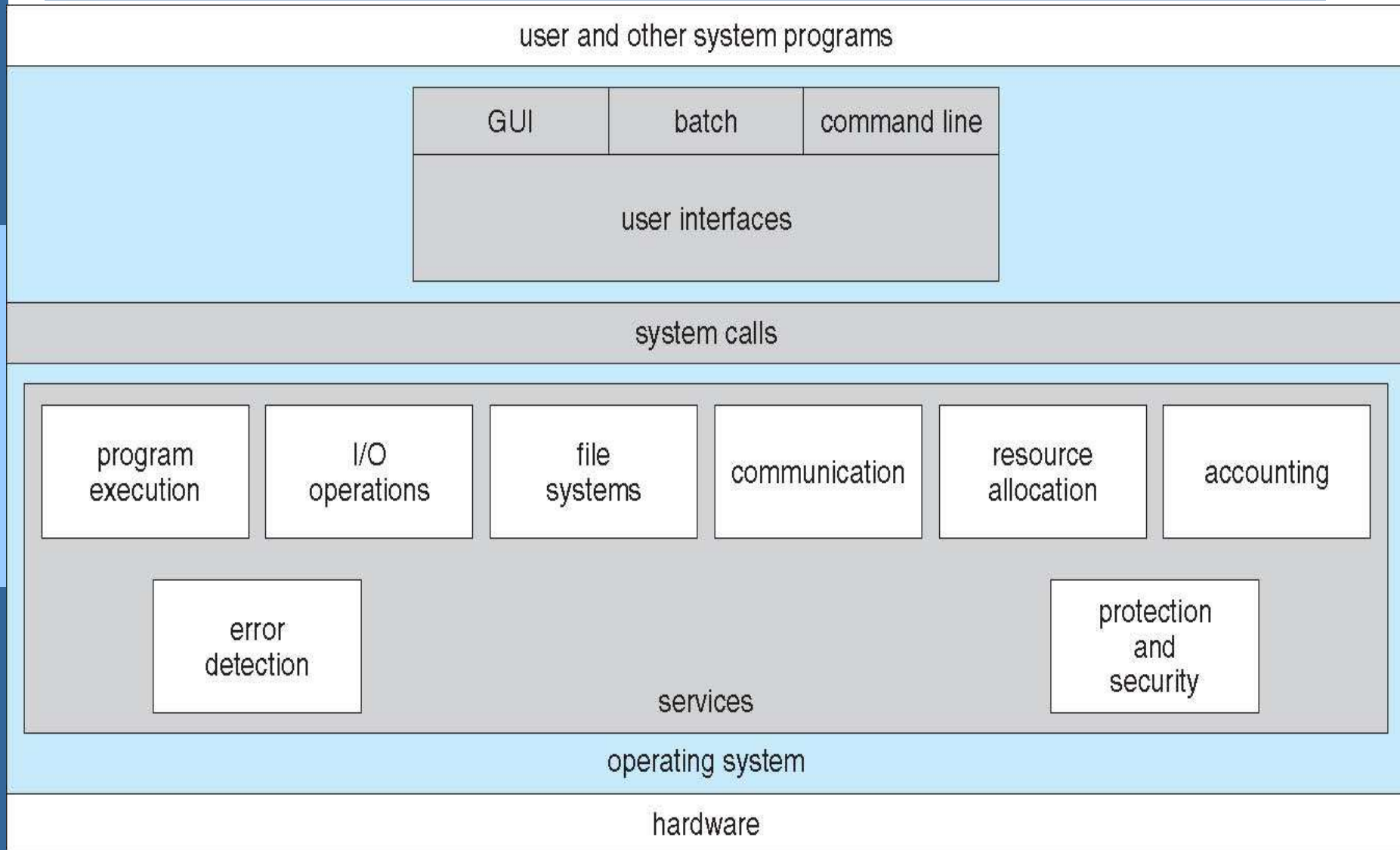
Operating System Services (Cont.)

- 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 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, extends to defending external I/O devices from invalid access attempts





A View of Operating System Services





Touchscreen Interfaces

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





The Mac OS X GUI





System Calls

- As we know that for performing any operation as user must have to specify the operation which he wants to operate on the computer.
- We can say that for performing any operation a user must have to request for a service from the system.
- For making any request a user will prepare a special call.
- The system call is the request for running any program and for performing any operation on the system.
- When a user first time starts the system then the **system is in the user mode and when he request for a service then the user mode will be converted into the kernel mode** which just listen the request of the user and process the request and display the results those are produced after the processing.
- When a user request for opening any folder or when a moves his mouse on the screen, then this is called as the system call which he is using for performing any operation.





Example of System Calls

- System calls (or syscalls) are function calls made from user space programs into the kernel requesting for some service.
- Examples: `Open()`, `read()`, `write()`, `fork()`, `lseek()`, `clone()`, `wait()`, etc..
- 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)





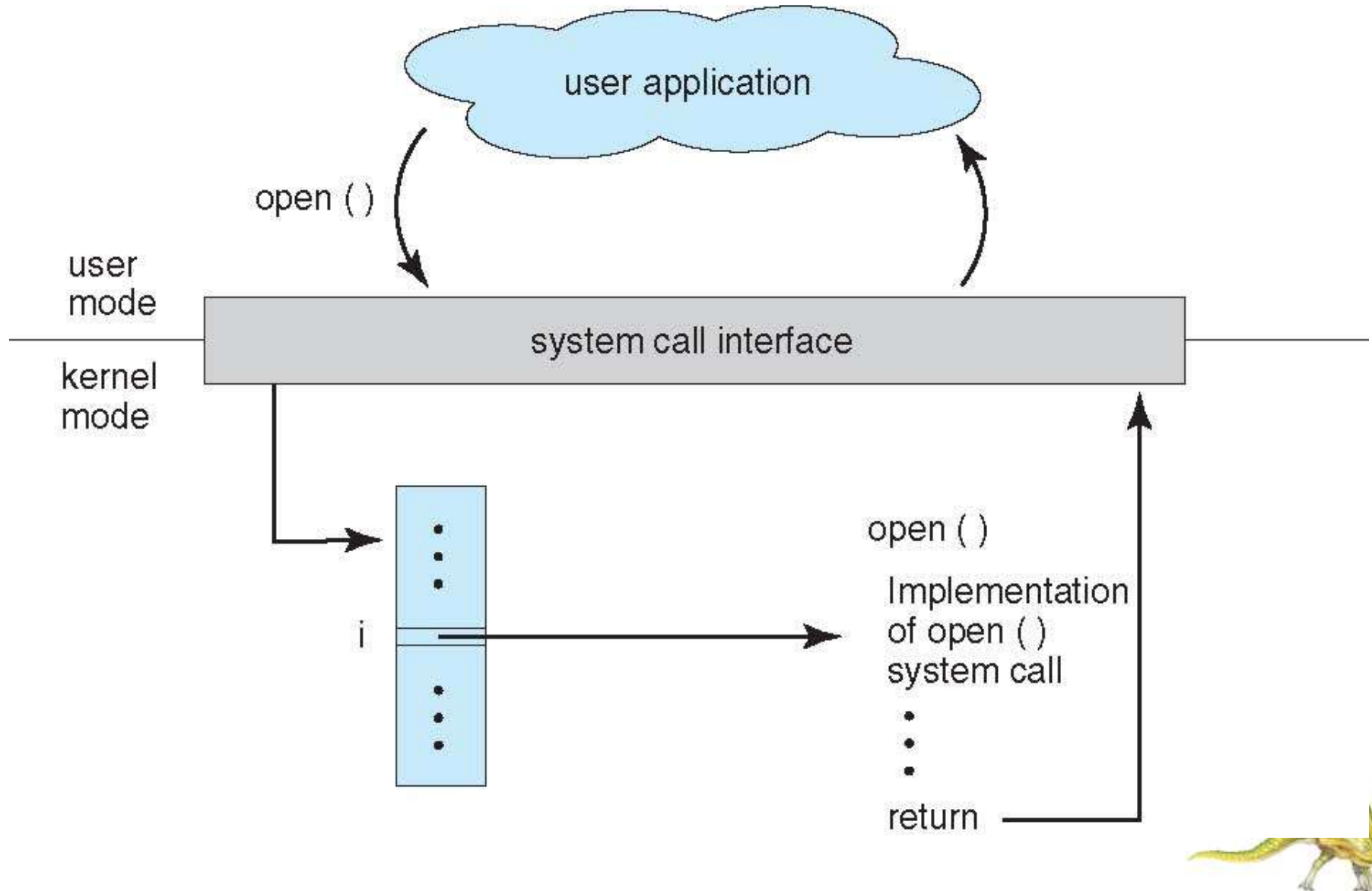
System Call Implementation

- Typically, a number associated with each system call
 - **System-call interface** maintains a table indexed according to these numbers
- The system call interface invokes the intended system call in OS kernel and returns status of the system call and any return values
- 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)





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, 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

■ Protection

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





Examples of Windows and Unix System Calls

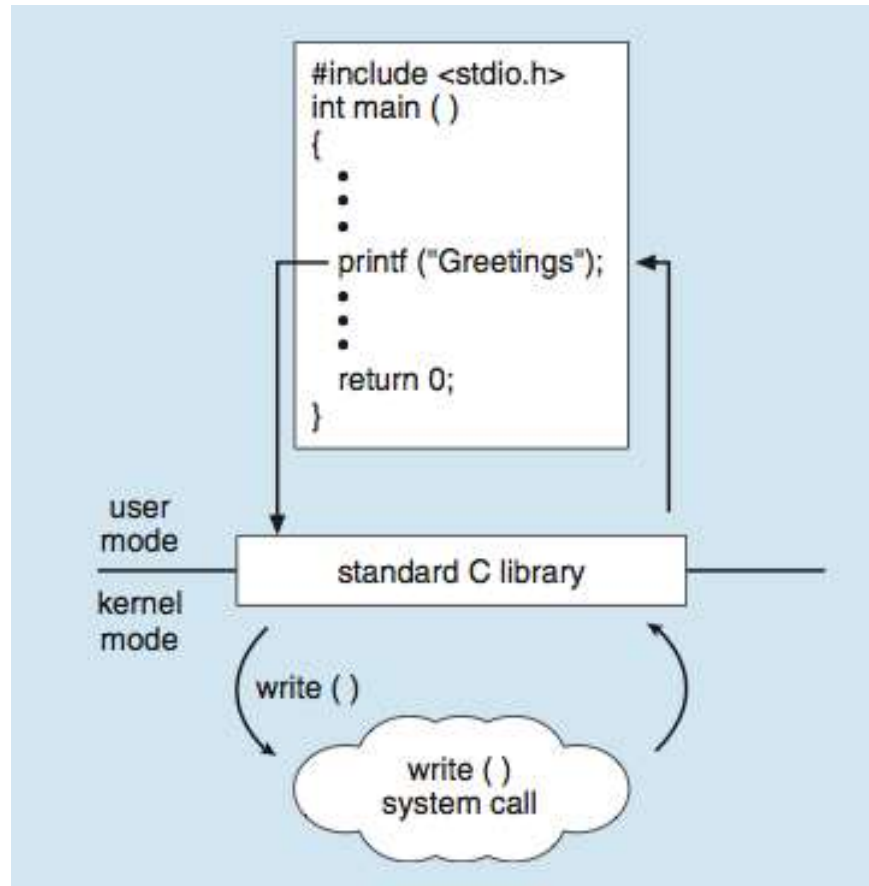
	Windows	Unix
Process Control	CreateProcess() ExitProcess() WaitForSingleObject()	fork() exit() wait()
File Manipulation	CreateFile() ReadFile() WriteFile() CloseHandle()	open() read() write() close()
Device Manipulation	SetConsoleMode() ReadConsole() WriteConsole()	ioctl() read() write()
Information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





Standard C Library Example

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





System Programs

- 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
- Most users' view of the operation system is defined by system programs, not the actual system calls





System Programs

- 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 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
 - Some systems implement a **registry** - used to store and retrieve configuration information





System Programs (Cont.)

- **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
- **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
 - 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 (Cont.)

■ 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
- Run in user context not kernel context
- Known as **services**, **subsystems**, **daemons**

■ Application programs

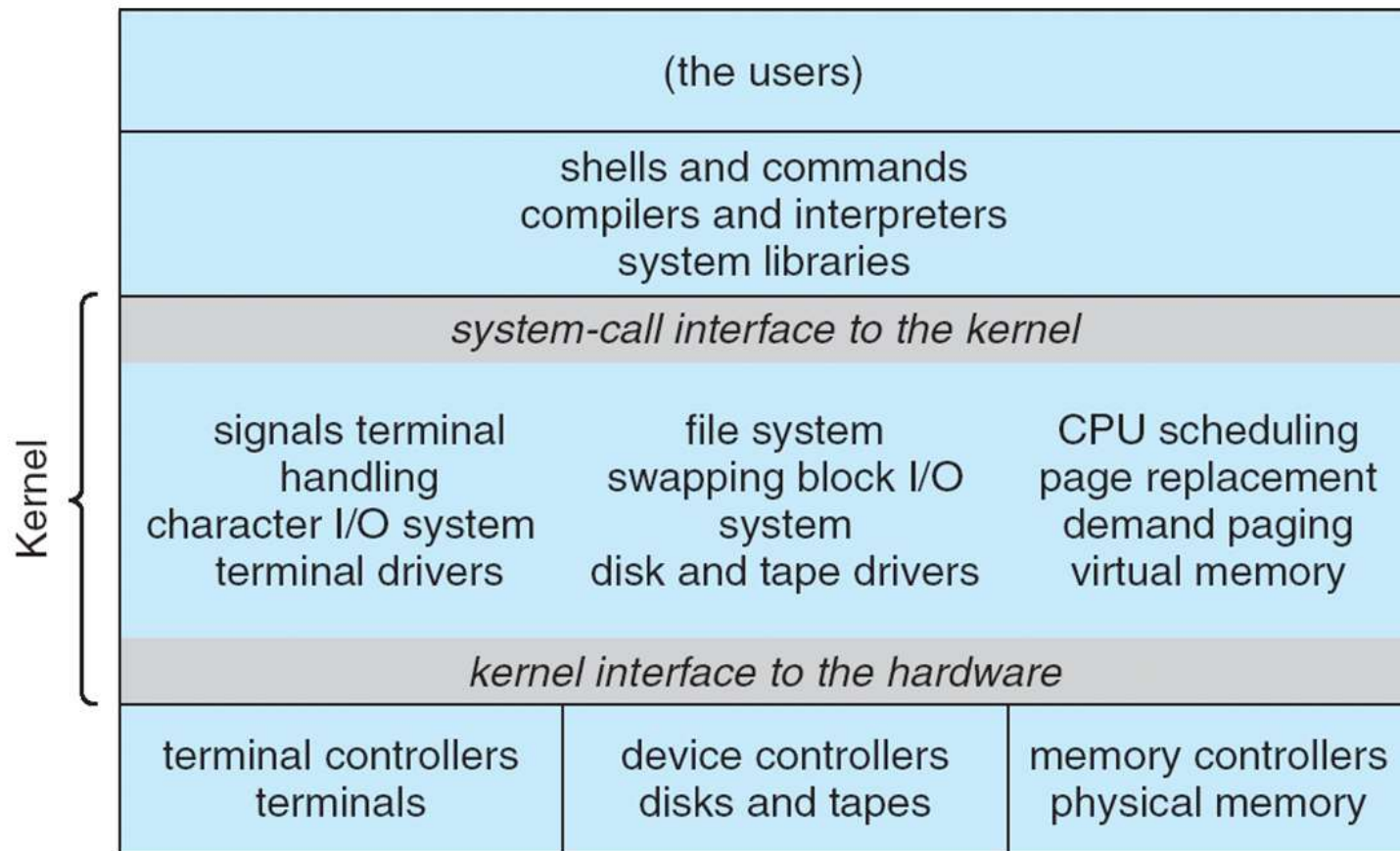
- Don't pertain to system
- Run by users
- Not typically considered part of OS
- Launched by command line, mouse click, finger poke





Traditional UNIX System Structure

Beyond simple but not fully layered





Operating-System Debugging

- **Debugging** is finding and fixing errors, or **bugs**
- OS generate **log files** containing error information
- Failure of an application can generate **core dump** file capturing memory of the process
- Operating system failure can generate **crash dump** file containing kernel memory
- Beyond crashes, performance tuning can optimize system performance
 - Sometimes using **trace listings** of activities, recorded for analysis
 - **Profiling** is periodic sampling of instruction pointer to look for statistical trends

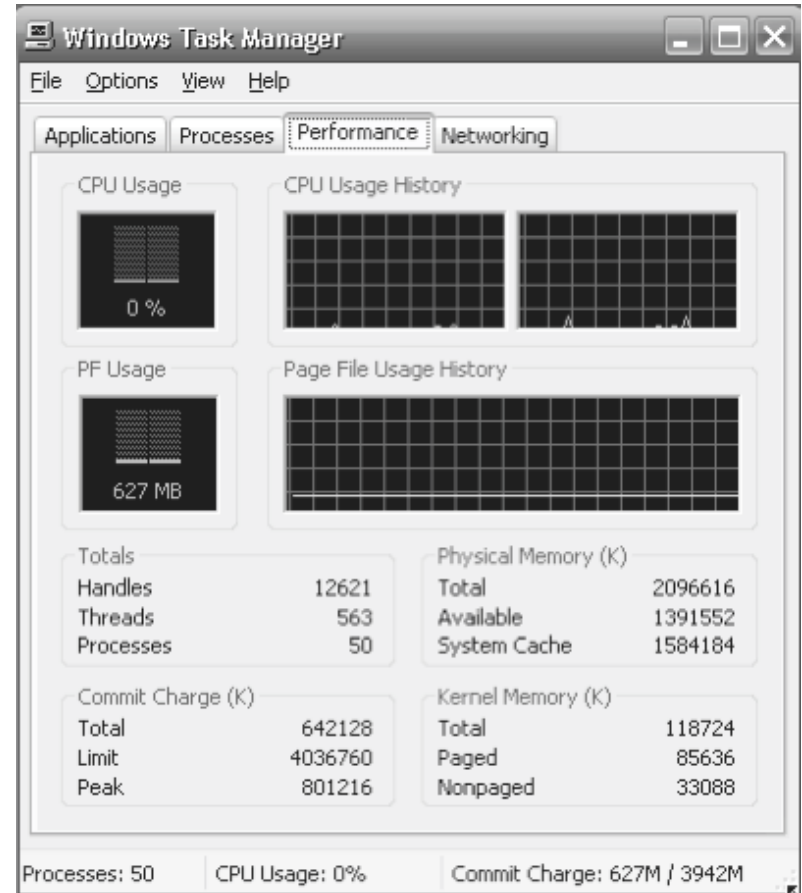
Kernighan's Law: "Debugging is twice as hard as writing the code in the first place. Therefore, if you write the code as cleverly as possible, you are, by definition, not smart enough to debug it."





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





System Boot

- When power initialized on system, execution starts at a fixed memory location
 - Firmware ROM used to hold 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 **boot block** at fixed location loaded by ROM code, which loads bootstrap loader from disk
- Common bootstrap loader, **GRUB**, allows selection of kernel from multiple disks, versions, kernel options
- Kernel loads and system is then **running**





Questions

- Describe OS services that are helpful to the user.
- Describe OS services that are helpful to OS itself.
- Define system calls with examples.
- Describe how system calls are implemented with a diagram.
- Give some example of system calls that are used in windows and UNIX OS.
- Make a list of some OS programs.
- Define operating system debugging.



End of Chapter 2

