Chapter 2

Operating System Services

Outline of this lecture

In this lecture, we will cover the following:

- 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

OS provides an environment for execution of programs and services to programs and users. One set of OS 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**)

- ***Program execution*** – system capability to load a program into memory and to run it. OS able to end a program's execution either normally or abnormally.

- ***I/O operations*** – since user programs cannot execute I/O operations directly, the operating system must provide some means to perform I/O.

- ***File-system manipulation*** – The OS gives the ability to read, write, create, open, close and delete files / directories.

- ***Error detection*** – ensure correct computing by detecting errors in the CPU and memory hardware, in I/O devices, or in user programs.

- ***Communications*** – exchange of information between processes executing either on the same computer or on different systems tied together by a network. Implemented via ***shared memory*** or ***message passing*** (packets moved by the OS, see chapter 3).

Additional Operating System Functions

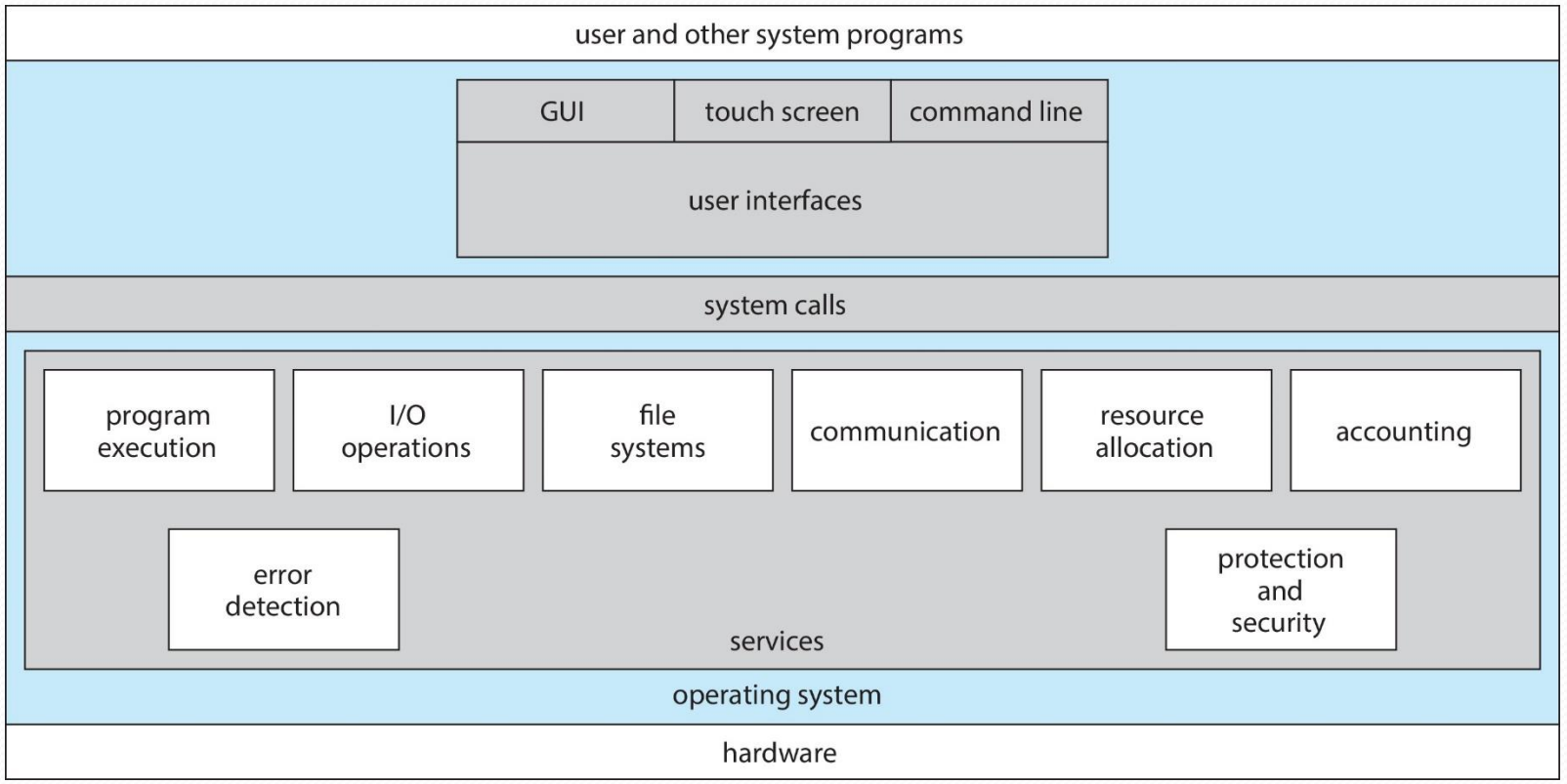
Additional functions exist not for helping the user, but rather for ensuring efficient system operations of the system itself via resource sharing.

- ***Resource allocation*** – allocating resources to multiple users or multiple jobs running concurrently, resources must be allocated to each of them.

- ***Accounting*** – keep track of and record which users use how much and what kinds of computer resources for account billing or for accumulating usage statistics.

- ***Protection & 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
  - If a system is to be protected and secure, precautions must be instituted throughout it. A chain is only as strong as its weakest link.

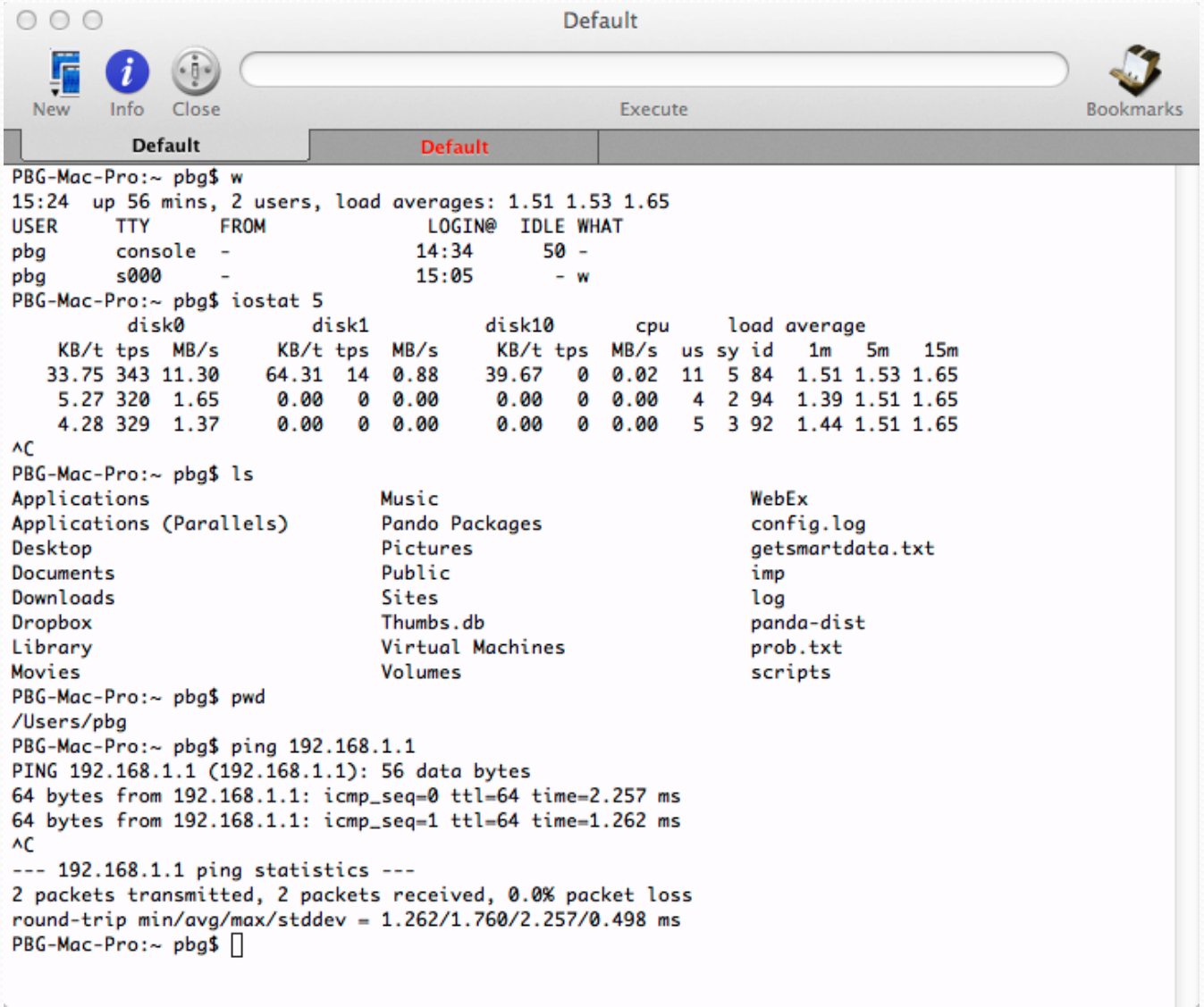
A View of Operating System Services



User Operating System Interface - CLI

- CLI or **command interpreter** allows direct command entry
  - Sometimes implemented in kernel, sometimes by systems program
  - On systems with multiple CLI to choose from, the interpreters are known – **shells**
  - Primarily fetches a command from user and executes it
    - Sometimes commands built-in, sometimes just names of programs
      - If the latter, adding new features doesn't require shell modification

Bourne Shell Command Interpreter- Solaris 10



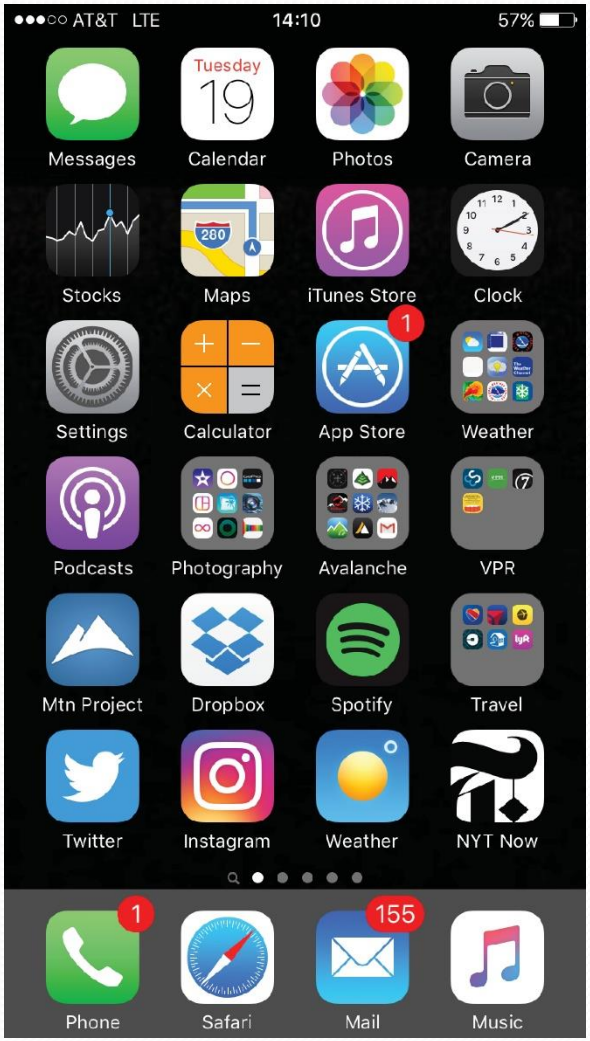

```
PBG-Mac-Pro:~ pbg$ w
15:24  up 56 mins, 2 users, load averages: 1.51 1.53 1.65
USER     TTY      FROM              LOGIN@  IDLE WHAT
pbg      console  -                 14:34      50 -
pbg      s000     -                 15:05       - w
PBG-Mac-Pro:~ pbg$ iostat 5
          disk0           disk1           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 - GUI

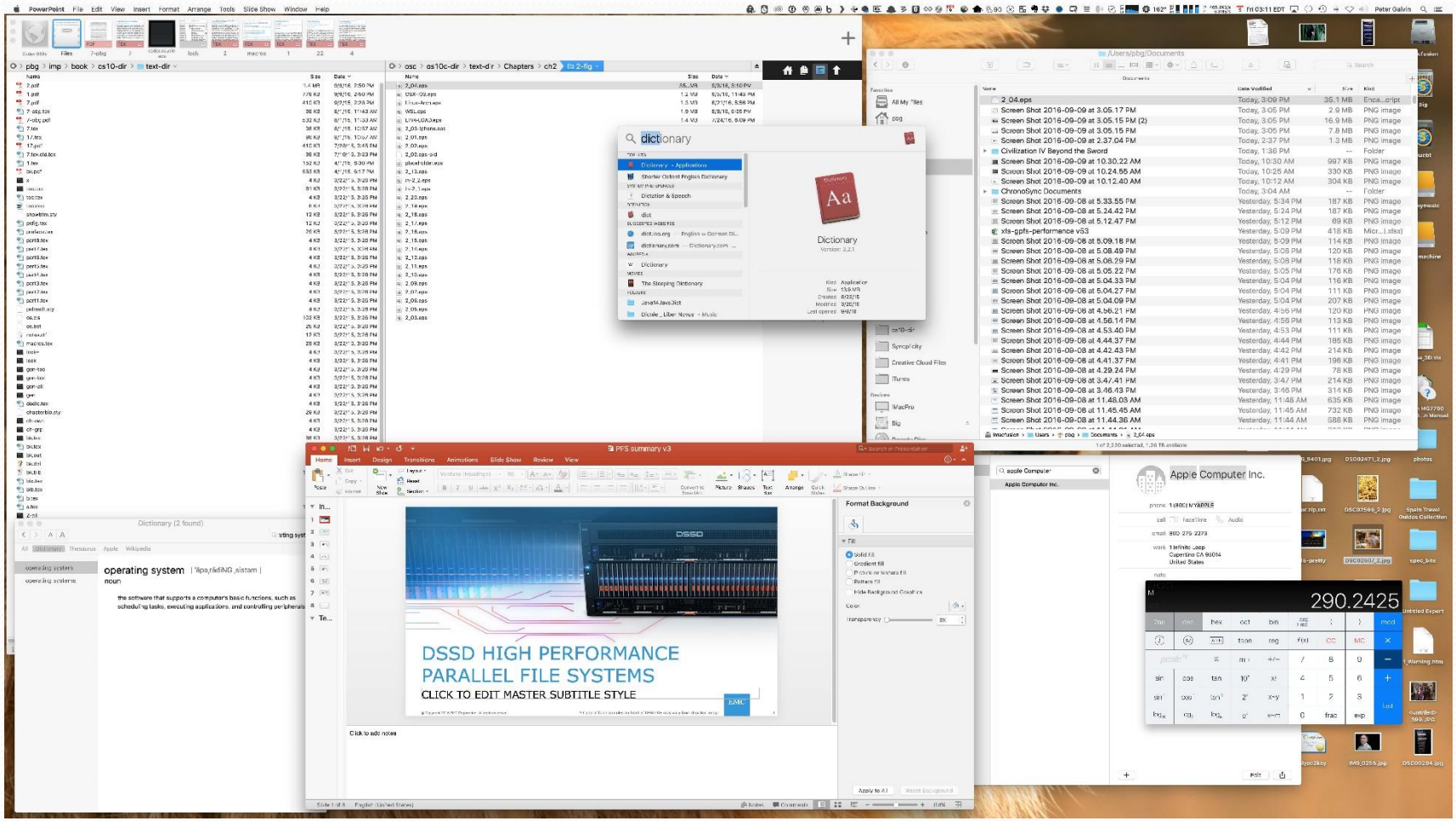
- User-friendly **desktop** metaphor interface
  - Usually mouse, keyboard, and monitor
  - **Icons** represent files, programs, actions, etc
  - Various mouse buttons over objects in the interface cause various actions (provide information, options, execute function, open directory (known as a **folder**)
  - Invented at Xerox PARC in the early 1970s
- 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)

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



The Mac OS X GUI



System Calls

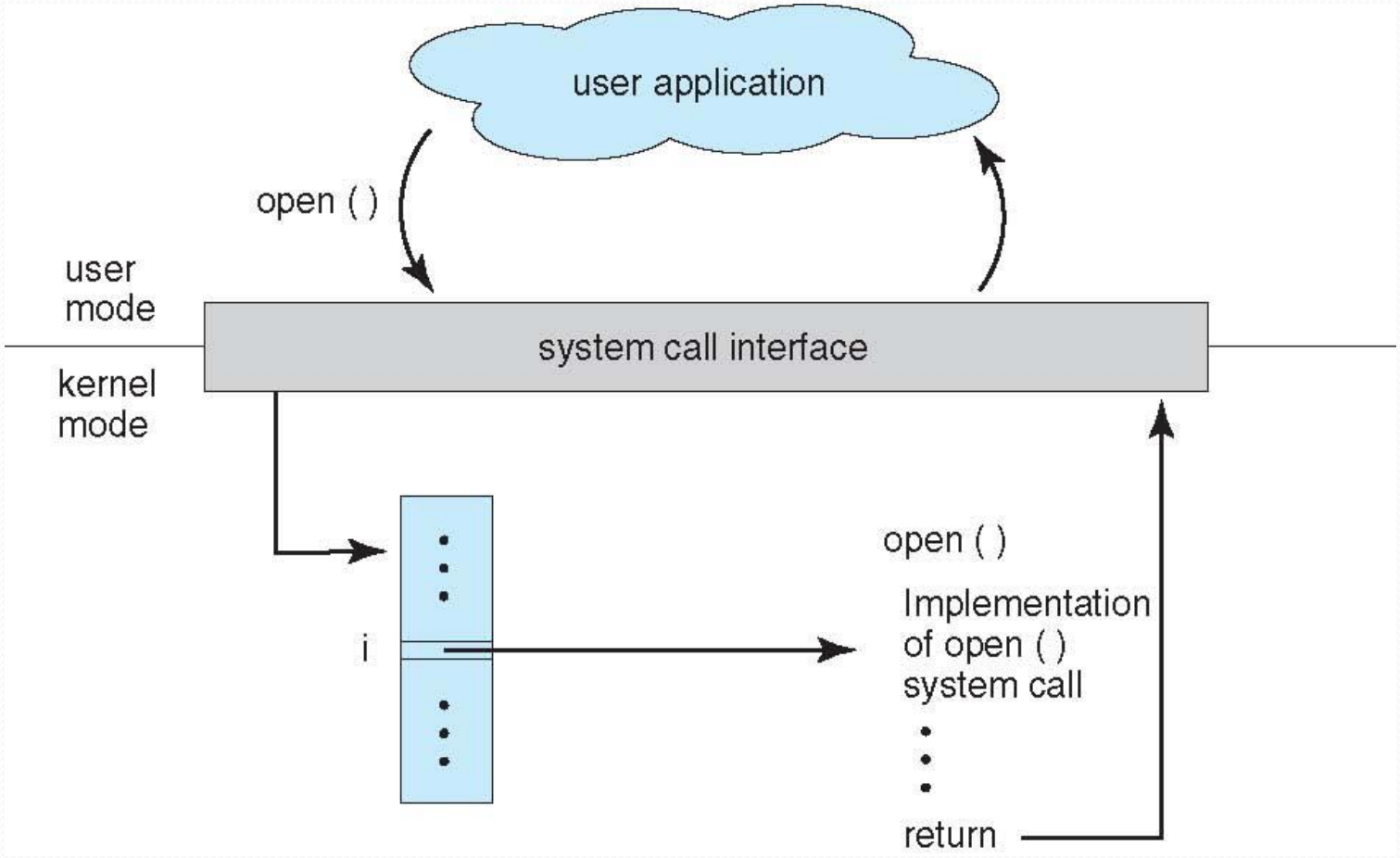
- System calls provide the interface between a running program and O/S.
- Typically written in a high-level language (C or C++)
- Mostly accessed by programs via a high-level **Application Program Interface** (**API**) rather than direct system call use
- Three most common APIs are Win32 API for Windows, 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)

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

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 intended system call in OS kernel and returns status of the system call and any return values
- The caller needs 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

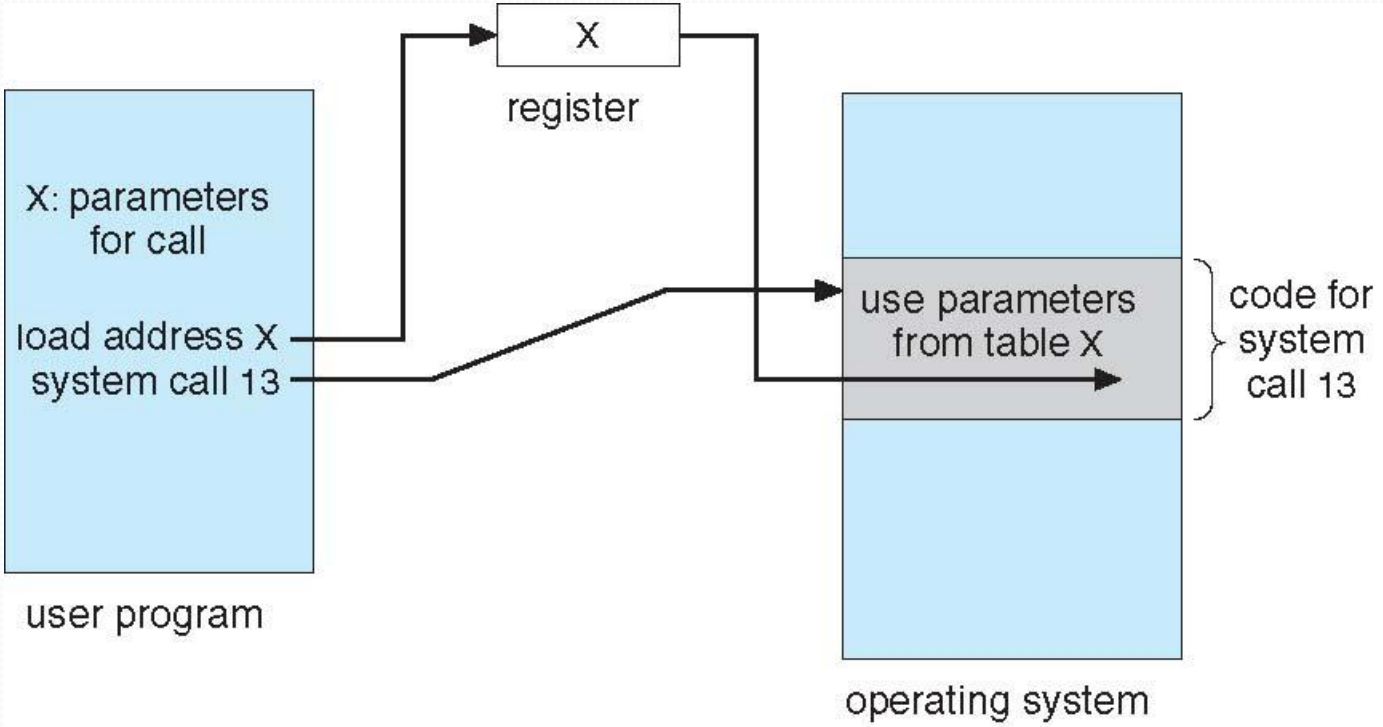


System Call Parameter Passing

- Often, more information is required than simply identity of desired system call
  - 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
    - In some cases, may be more parameters than 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 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

Passing of Parameters As A Table



Types of System Calls

System Calls can be grouped into 6 categories

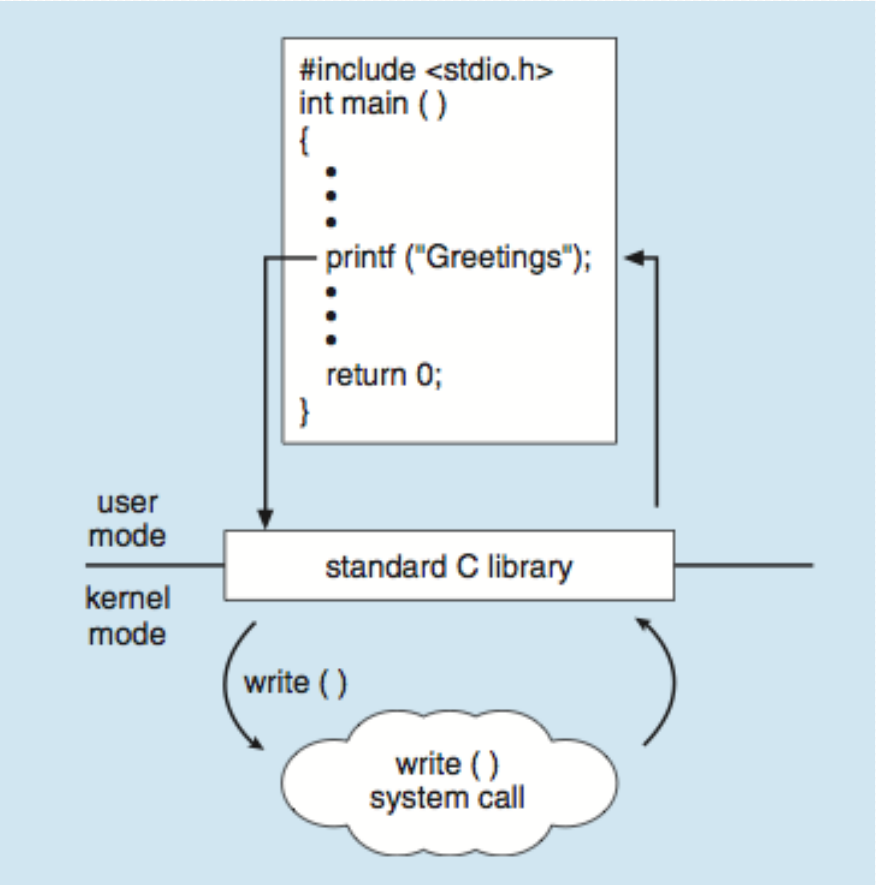
- **Process control** – create/terminate process, load/execute, end/abort, etc.
- **File management-** create/delete file, open/close, read/write, get/set file attributes
- **Device management**- read/write, get/set device attributes, request/release device.
- **Information maintenance-** get/set time or date, get/set file attributes, get/set process/device attributes.
- **Protection** – Control access to resources, Get and set permissions, Allow and deny user access
- **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

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() InitlializeSecurityDescriptor() SetSecurityDescriptorGroup()	chmod() umask() chown()

Standard C Library Example

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



Operating System Structure

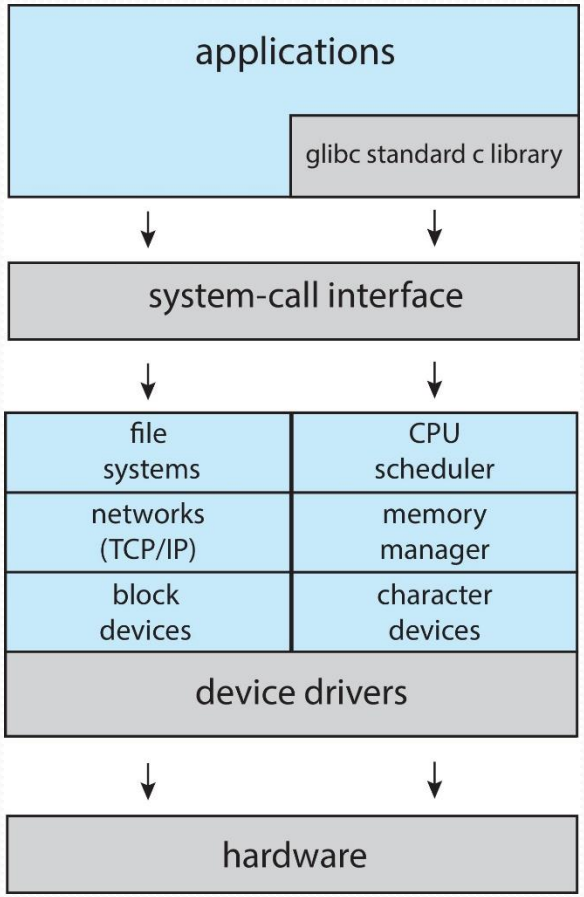
- General-purpose OS is very large program
- Various ways to structure ones
  - Simple structure – MS-DOS
  - More complex – UNIX
  - Layered – an abstraction
  - Microkernel – Mach

Linux System Structure

Monolithic plus modular design

Monolithic : all the functionality of the kernel placed into a single, static binary file that runs in a single address space.

changes to one part of the system can have wide-ranging effects on other parts.

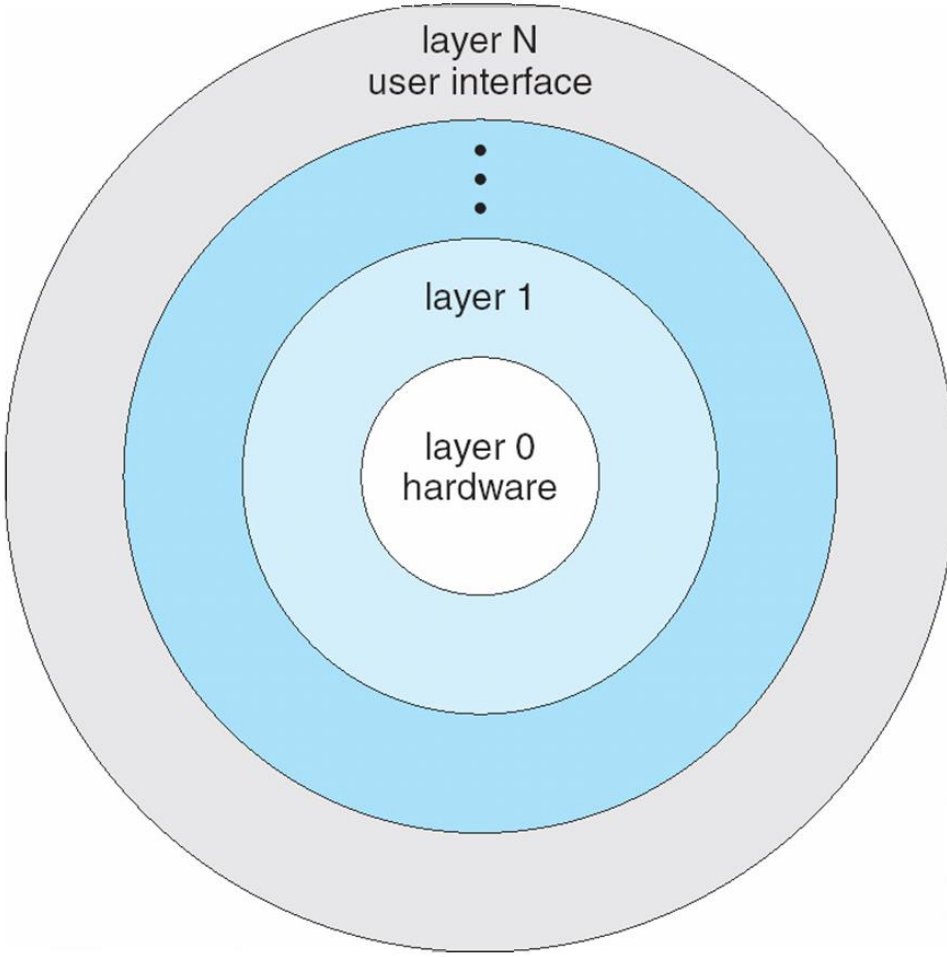


2. Layered Approach

- The operating system is divided into a number of layers (levels), each built on top of lower layers. The bottom layer (layer o), is the hardware; the highest (layer N) is the user interface.

The system is divided into separate, smaller components that have specific and limited functionality.

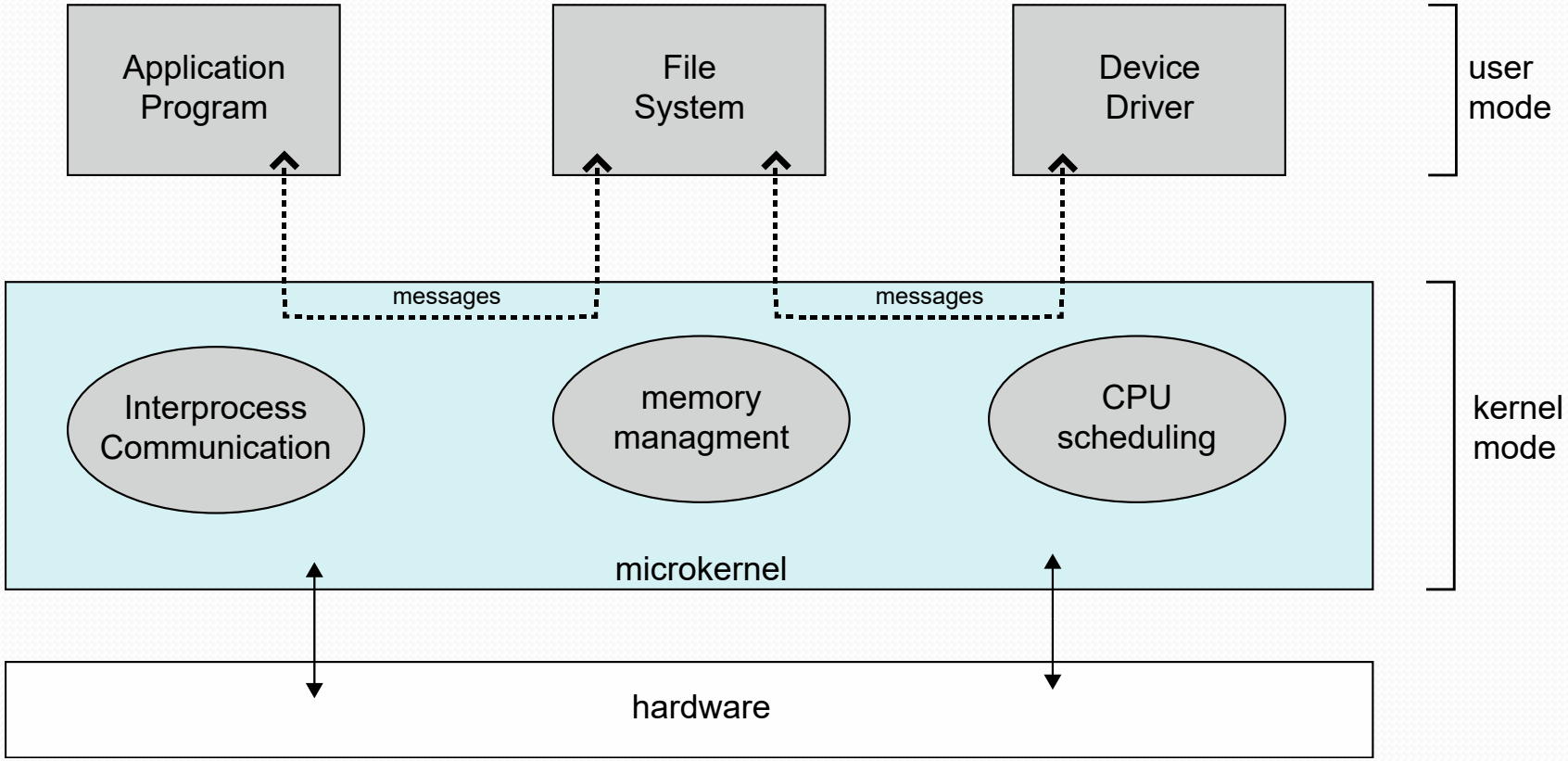
- With modularity, layers are selected such that each uses functions (operations) and services of only lower-level layers



3. Microkernel System Structure

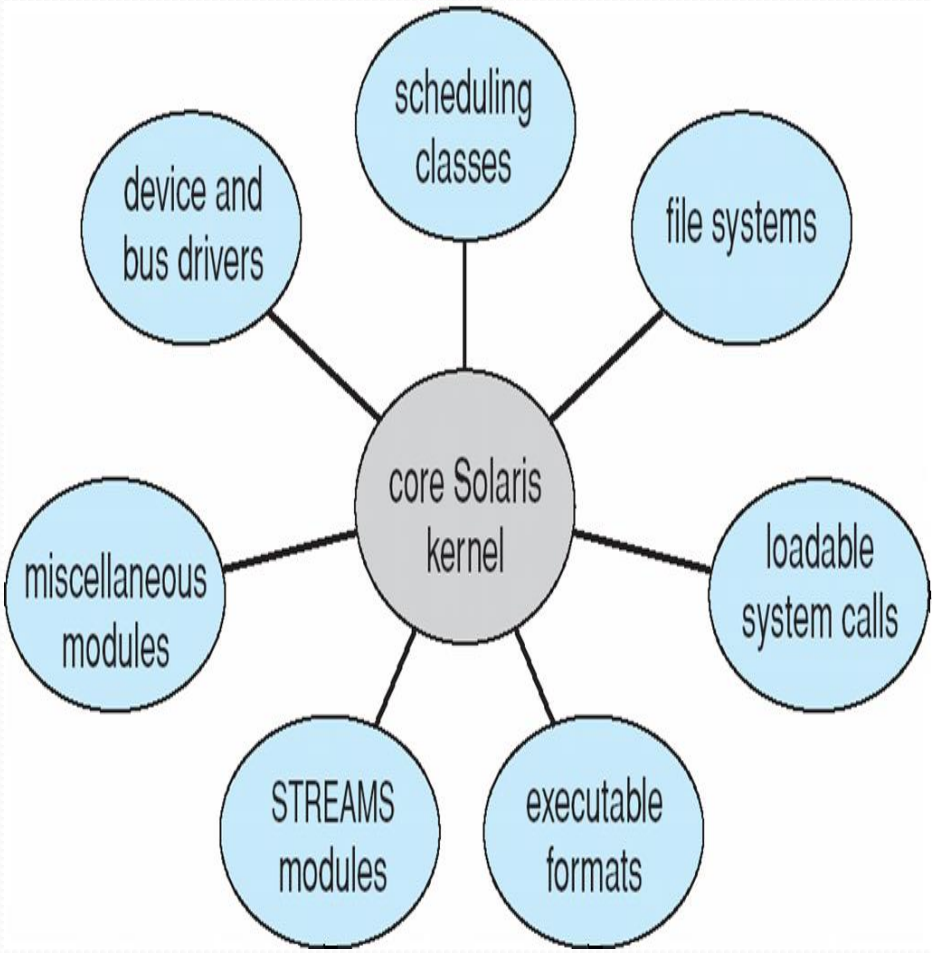
- Removes all nonessential components from the kernel & implementing them as system & user-level progs- smaller kernel. E.g. Mach OS
- Microkernel provides communication between client prog & various services that are running in user space using *message passing*.
- Benefits:
  - extension of the OS is easier when there is less kernel to modify.
  - easier to port the operating system to a different hardware platform because there's less kernel to reimplement.
  - more reliable (less code is running in kernel mode).
  - more secure (most services are running as user rather than kernel)

Microkernel System Structure



Modules

- Most modern operating systems implement **loadable kernel modules**
- the kernel has a set of core components and can link in additional services via modules, either at boot time or during run time. This type of design is common in modern implementations of UNIX, such as Linux, macOS, and Solaris, as well as Windows.
  - Uses object-oriented approach
  - Each core component is separate
  - Each talks to the others over known interfaces
  - Each is loadable as needed within the kernel
- Overall, similar to layers but more flexible because any module can call any other module



Hybrid Systems

- Most modern operating systems actually not one pure model
  - Hybrid combines multiple approaches to address performance, security, usability needs
  - Linux and Solaris kernels in kernel address space, so monolithic, plus modular for dynamic loading of functionality
- Windows mostly monolithic, plus microkernel providing support for separate subsystems that run as user-mode processes.

Operating-System Debugging

- **Debugging** is finding and fixing errors, or **bugs**
- OSes 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