


Operating System Structure

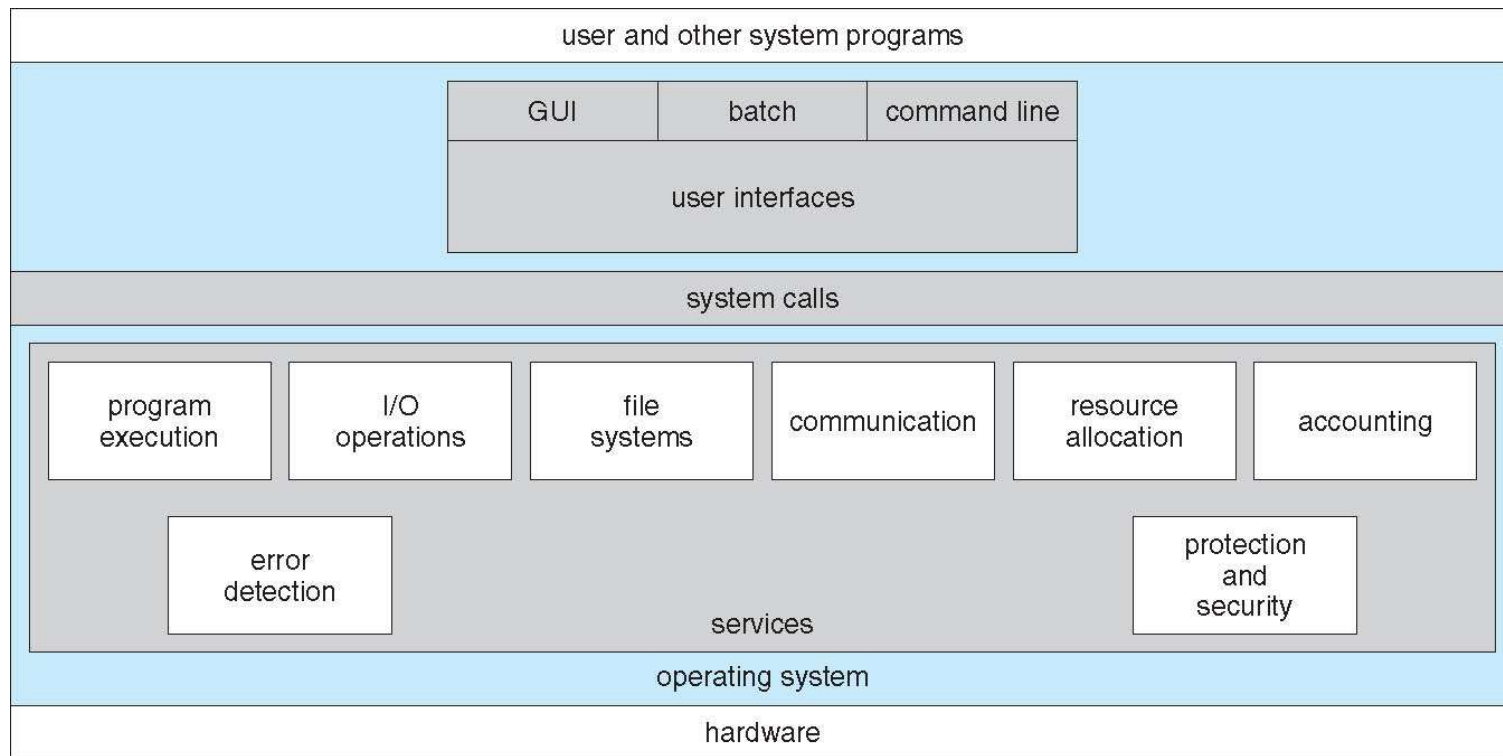
CHAPTER 2



Outline

- Operating system services
 - User and operating system-interface
 - System calls
 - System programs
 - Operating system structures
- 

OS Services



OS Services

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

➤ Services provided by the OS can be:

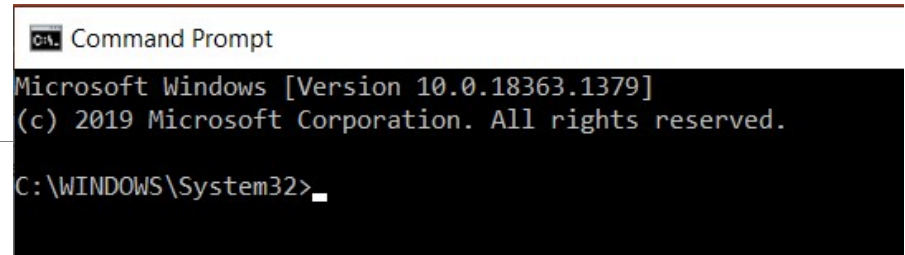
- **User interface** - Command-Line (CLI), Batch, Graphics User Interface (GUI)
- **Program execution** - Load a program into memory and run that program
- **I/O operations** - Interaction with I/O devices
- **File-system manipulation** - Aids programs to read and write files and directories
- **Communications** – Helps processes exchange information with each other
- **Error detection** – Detects and corrects errors for reliable computing

➤ Some other Services provided by the OS for efficient operations can be :

- **Resource allocation** – Allocates resources to multiple jobs, e.g., CPU cycles, I/O devices
- **Accounting** – Tracks usage of resources for billing users for resource usage purposes
- **Protection and security** – Protect and secure multiuser environment
 - 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

OS Interfaces

CLI (Command Line Interface)



```
Command Prompt
Microsoft Windows [Version 10.0.18363.1379]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\WINDOWS\System32>
```

- CLI or command interpreter allows direct command entry
 - Sometimes implemented in kernel, sometimes by systems program
 - Can have multiple flavors implemented – shells, e.g., C Shell, Bourne Shell. A shell is a user interface for access to an operating system's services.
 - Works in two ways
 - Fetches a command from user and executes it
 - loads a program in the memory and executes it

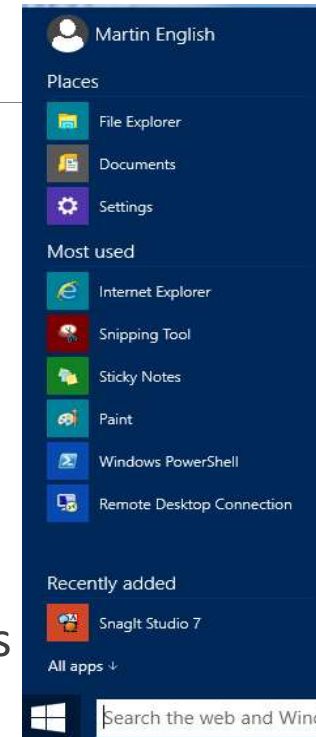
rm file.txt

OS Interfaces

GUI (Graphical User Interface)

- User-friendly desktop interface
 - Usually mouse, keyboard, and monitor
 - Icons represent files, programs, actions, etc.
 - Invented at Xerox PARC
- 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
 - Unix and Linux have CLI with optional GUI interfaces (CDE, KDE, GNOME)

Windows



Ubuntu



GENOME

Mac OS X



OS Interfaces (GUI)

Touch Interfaces

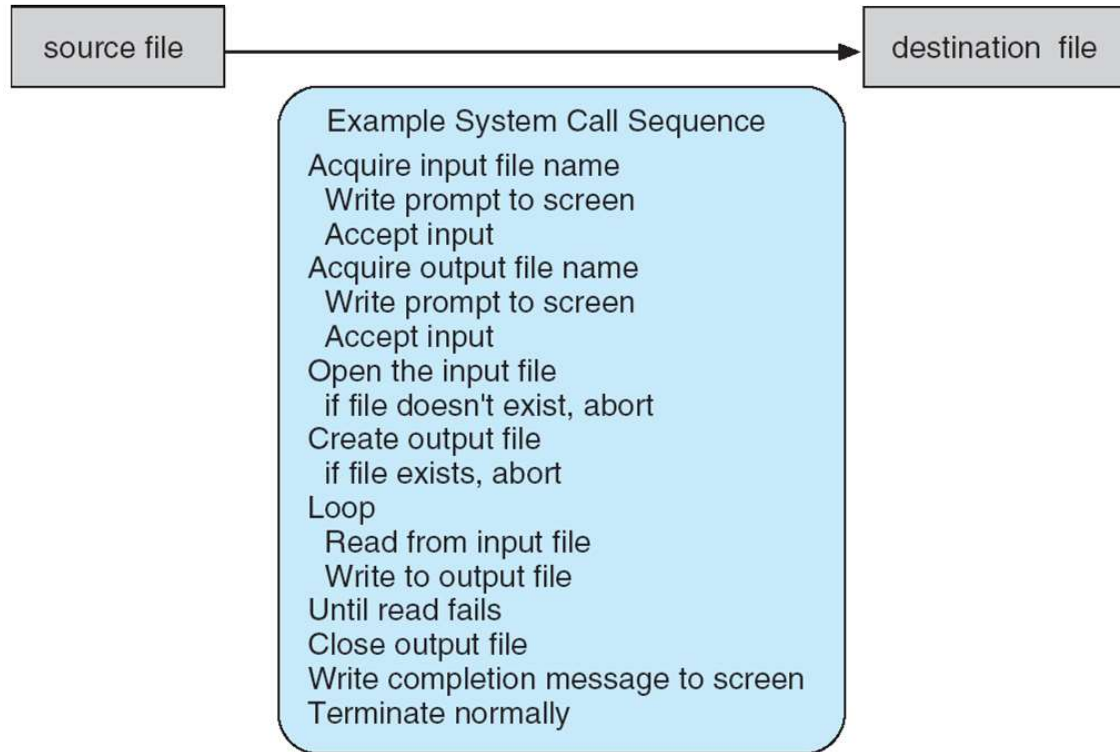
...



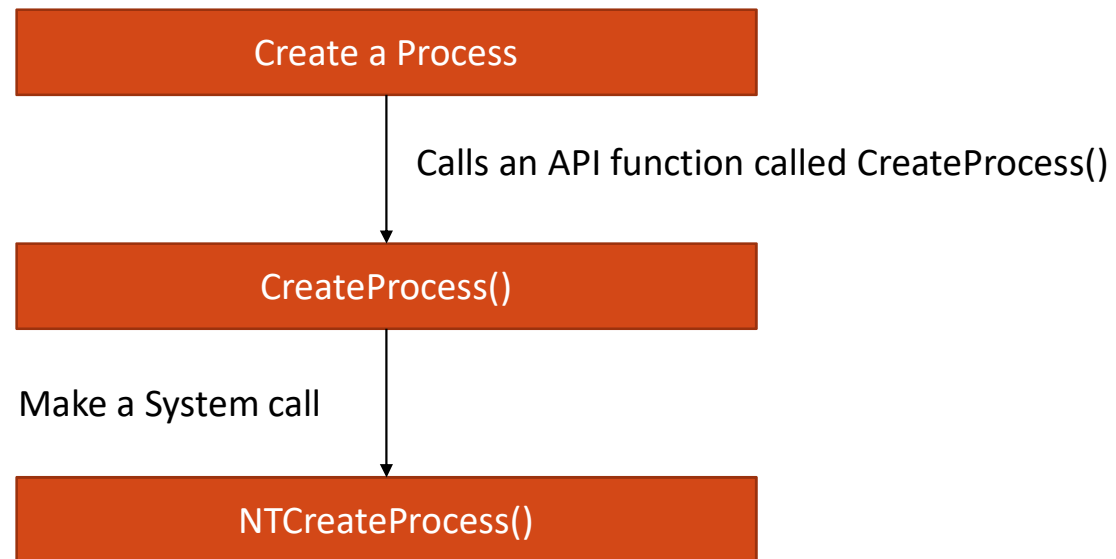
System Calls

- 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
- Three most common APIs are
 - Win32 API for Windows
 - POSIX API for POSIX-based systems
 - Java API for the Java virtual machine (JVM)

System Call for Copying a File Content



System Call for Creating a Process



Standard API Example

- Below is an example of a standard API for a `read()` function in UNIX and Linux
- 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`
- On Unix-like systems, **`unistd.h`** is made up of system call wrapper functions [API] such as `read` and `write`.
- File Descriptor or **`fd`** is an integer number that uniquely represents an opened file in operating system.

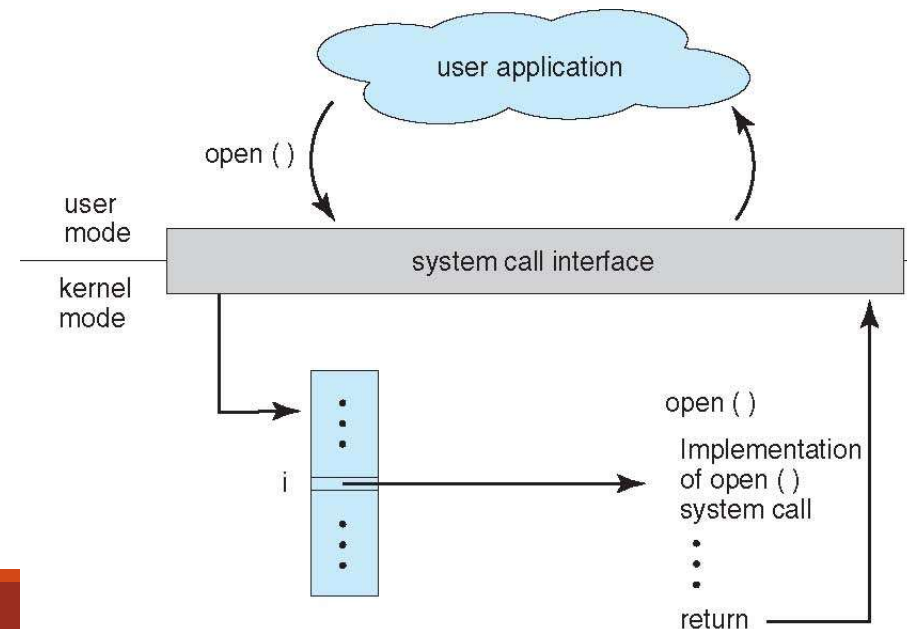
```
#include <unistd.h>

ssize_t read(int fd, void *buf, size_t count)
```

The diagram illustrates the components of the `read()` function signature. It shows the return type `ssize_t`, the function name `read`, and the parameters `(int fd, void *buf, size_t count)`. Brackets are used to group these elements and label them as 'return value', 'function name', and 'parameters' respectively.

System Call Implementation

- 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 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 are hidden from the programmer by API



Types of System Calls

- There are different types of System Calls
 - Process Control
 - File Manipulation
 - Device Manipulation
 - Information Maintenance
 - Communications
 - Protection

	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()

Windows and Unix APIs for System Calls

Types of System Calls

Process Control

- create and terminate process
- end, abort
- load, execute
- get and set process attributes
- wait for time
- wait event, signal event
- allocate and free memory
- dump memory if error

File Management

- create file/directory
- delete
- read
- write
- close
- move
- copy

Types of System Calls

Device Management

- request a resource (device)
 - These devices can be physical like disk, memory or virtual like file
 - If the device is not available, the process has to wait
- release a resource (device)
 - Once the process is done working with the device, it should release it so others can use it
- read
- write

Information Maintenance

- getting the number of users
- time
- date
- operating system version
- amount of free disk space
- getting memory dumps

Types of System Calls

Communication

- getting host id
- getting process id
- open connection
- close connection
- accept connection
- read message
- write message

Protection

- set permission
- get permission
- allow user
- deny user

System Programs

- System programs provide a convenient environment for program development and execution.
- Most users' view of the operation system is defined by system programs, not the actual system calls

File manipulation

- Create, delete, copy, rename, print, list, and 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

File modification

- Text editors to create and modify files
- Special commands to search the contents of files or perform transformations of the text

Programming-language support

- Compilers, assemblers, debuggers and interpreters are sometimes provided

Program loading and execution-

- loaders, relocatable loaders, debugging systems, etc

System Programs


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


Background Services

- Launch at boot time
- Provide facilities like disk checking, process scheduling, error logging, printing
- Run in user context not kernel context
- Known as services, subsystems, daemons

Application programs

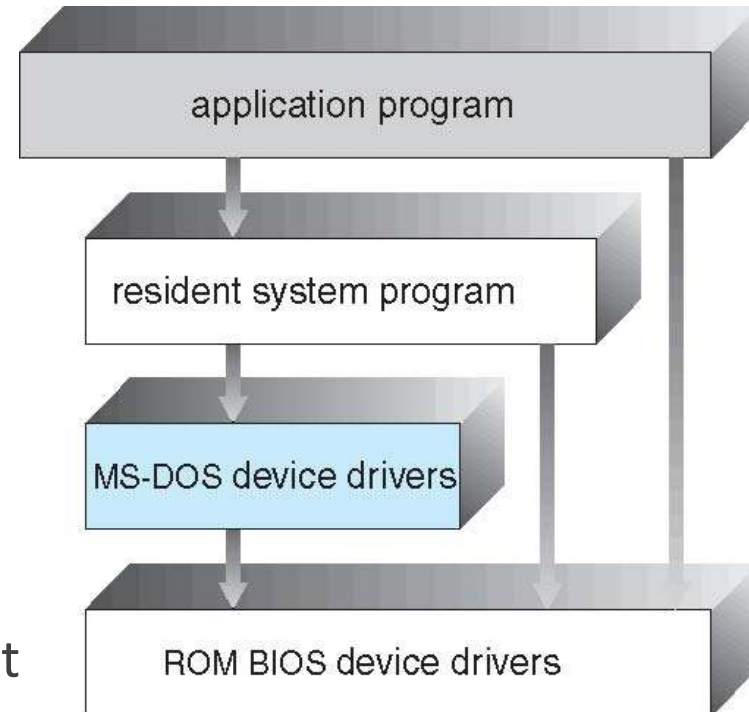
- Don't pertain to system
 - Run by users
 - Not typically considered part of OS
 - Launched by command line, mouse click, finger poke
- 

Operating System Structure

- General-purpose OS is very large program
 - Various ways to structure ones
 - Simple structure
 - Layered
 - Microkernel
 - Modules
 - Hybrid
- 

Operating System Structure: Simple Structure

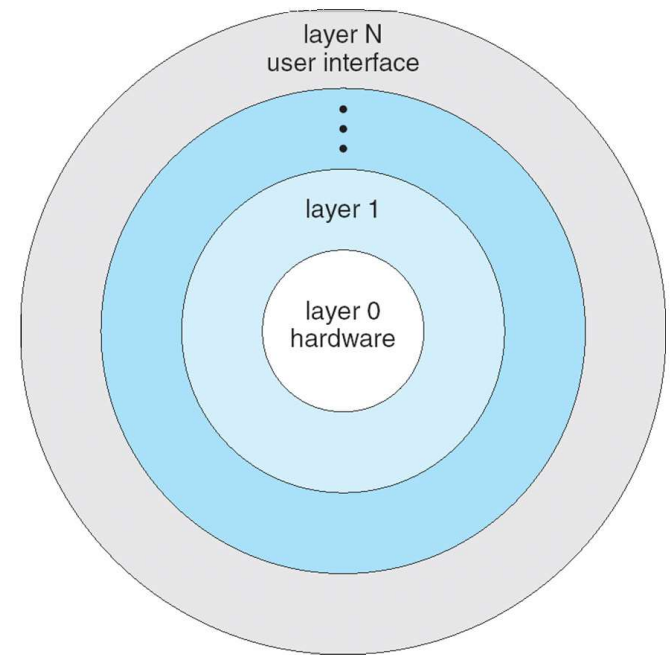
- MS-DOS – written to provide the most functionality in the least space
- Not divided into modules
- interfaces and levels of functionality are not well separated
- Apple PRODOS, and Apple Macintosh, are **single-tasking**.
- OS/2 and different versions of Unix and Microsoft Windows are **multitasking**.
- **resident program:** It is a program that is always present in the computer's memory, thus being "resident".



Operating System Structure: Layered structure

Windows NT

- 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



Operating System Structure: Microkernel structure

Minix 3 OS - Unix-like operating system.

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

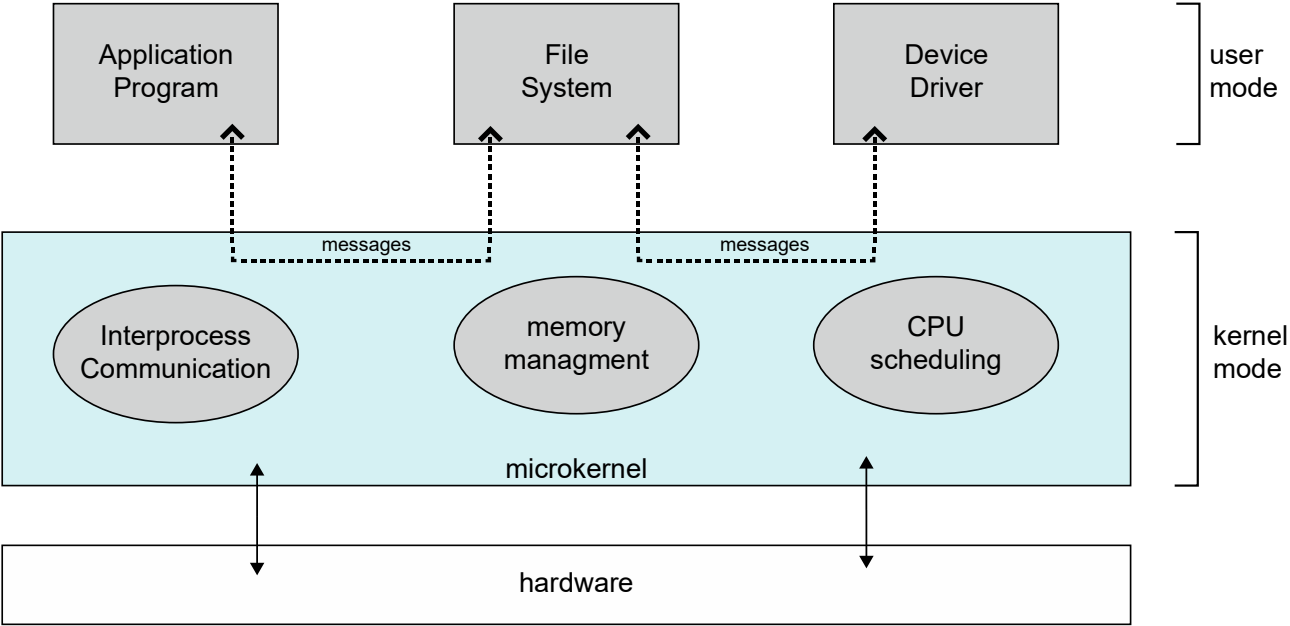
Benefits:

- Easier to extend a microkernel
- Easier to port the operating system to new architectures
- More reliable
- More secure

Detriments:

- Performance overhead of user space to kernel space communication

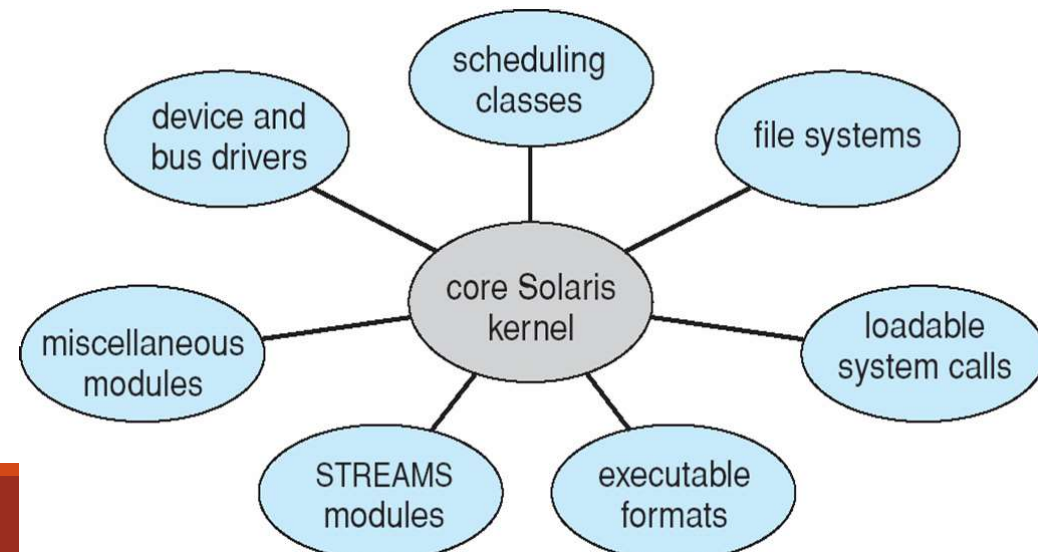
Operating System Structure: Microkernel structure



Operating System Structure: Modules

- Many modern operating systems implement loadable kernel modules
- 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
- Linux, Solaris, etc.

Solaris Modular Approach



Operating System Structure: Hybrid

- Most modern operating systems are actually not one pure model
 - Hybrid combines multiple approaches to address performance, security, usability
 - Linux and Solaris kernels are monolithic, plus modular for dynamic loading of functionality
 - Windows is mostly monolithic, plus microkernels for different subsystem personalities
 - Apple Mac OS X hybrid, layered, Aqua UI plus Cocoa programming environment

- Mach: For parallel programming

Mac OS X structure

