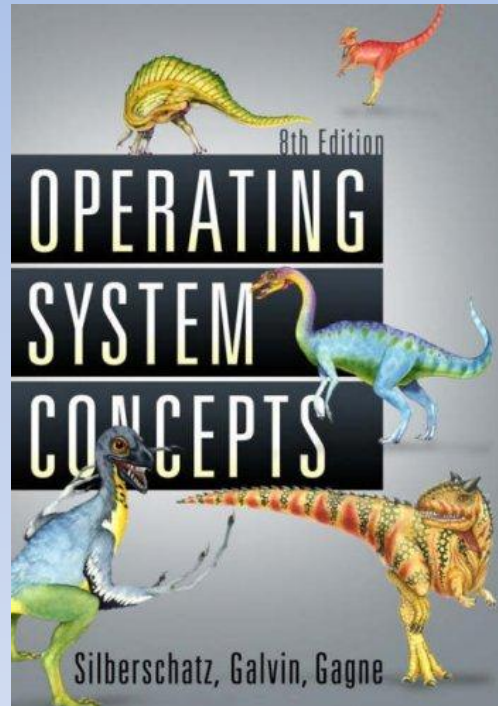


# Introduction to PC Operating Systems



***Operating System Concepts 8th Edition***  
***Written by: Abraham Silberschatz, Peter Baer Galvin and Greg Gagne***

***John Wiley & Sons, Inc.***  
***ISBN: 978-0-470-12872-5***

# Chapter 2

## Operating-System Structure

The design of a new operating system is a major task. It is important that the goals of the system be well defined before the design begins. These goals form the basis for choices among various algorithms and strategies.

What can be focused on in designing an operating system

- Services that the system provides
- The interface that it makes available to users and programmers
- Its components and their interconnections

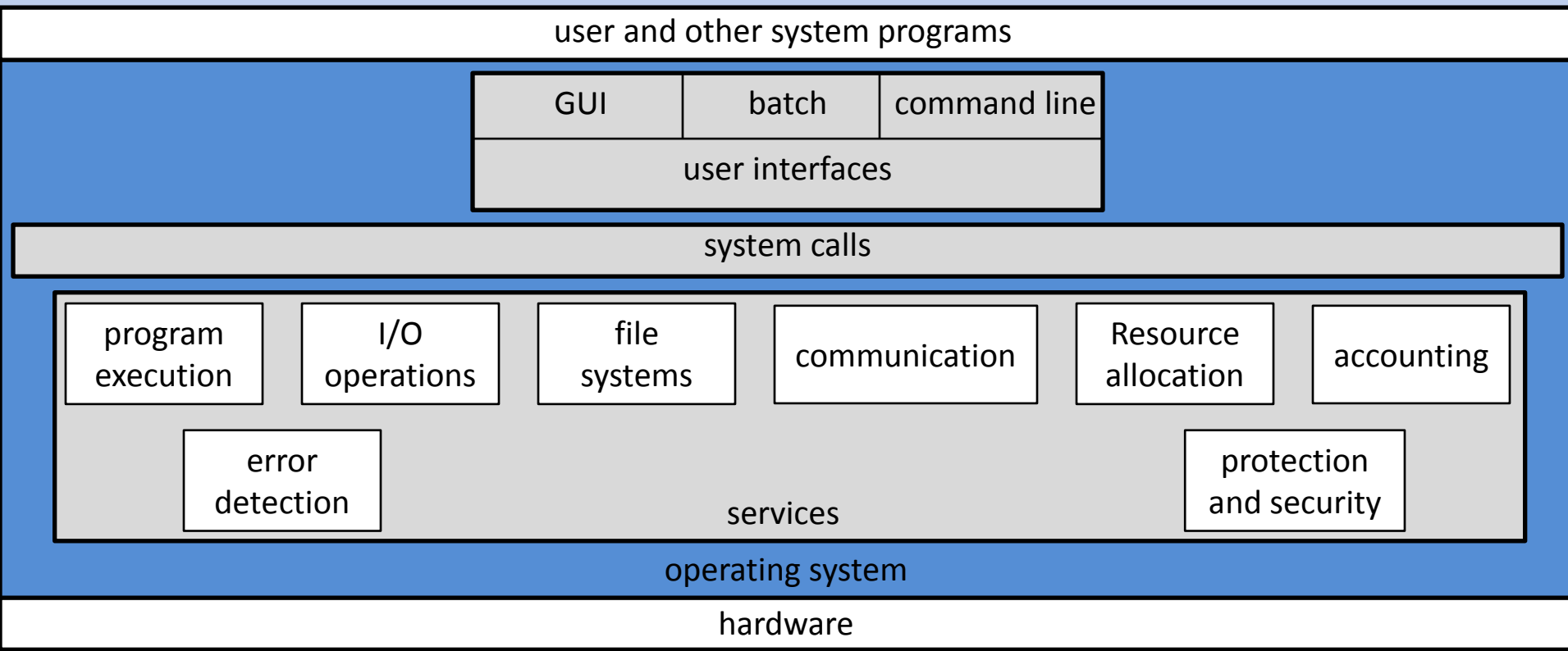
# Operating-System Services

The operating system provides:

- **An environment for the execution of programs**
- Certain services to programs and to the users of those programs.

Note: (services can differ from one operating system to another)

## A view of operating system services

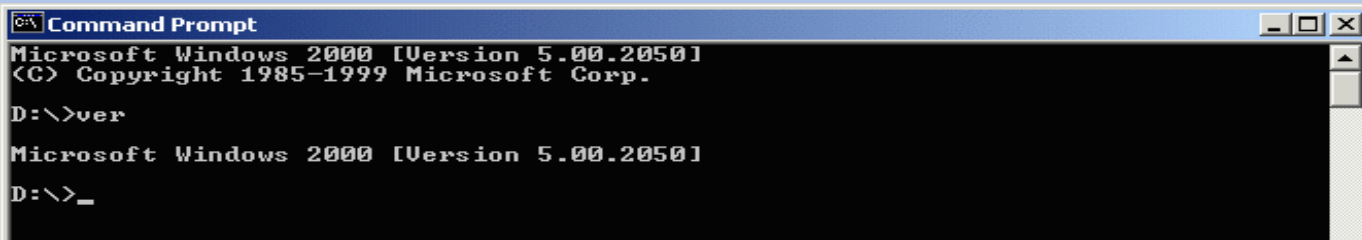


# Operating-System Services

Almost all operating systems have a user interface.

**Types of user interfaces:**

1. **Command Line Interface** (CLI) – method where user enter text commands



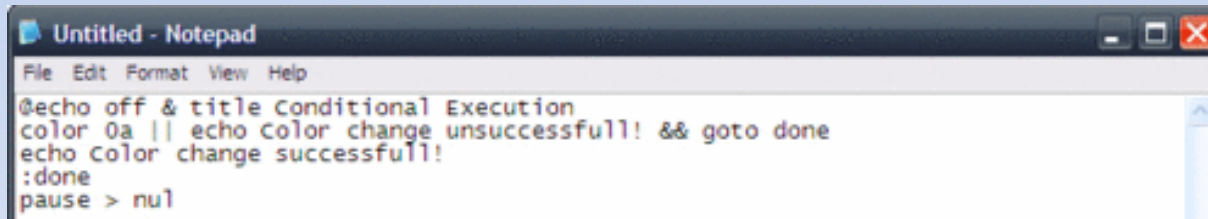
```
Command Prompt
Microsoft Windows 2000 [Version 5.00.20501]
(C) Copyright 1985-1999 Microsoft Corp.

D:\>ver

Microsoft Windows 2000 [Version 5.00.20501]

D:\>_
```

2. **Batch Interface** – commands and directives that controls those commands are within a file and the files are executed



```
Untitled - Notepad
File Edit Format View Help
@echo off & title Conditional Execution
color 0a || echo color change unsuccessful! && goto done
echo color change successful!
:done
pause > nul
```

3. **Graphical User Interface** – works in a windows system using a pointing device to direct I/O, using menus and icons and a keyboard to enter text.



# Operating-System Services

## Services

**Program Execution** – the system must be able to load a program into memory and run that program. The program must be able to end its execution, either normally or abnormally (indicating error).

**I/O Operations** – this operation may involve files and I/O devices. (i.e. recording to CD or DVD or blanking a screen) For efficiency and protection, users usually cannot control I/O devices. Therefore, the operating system must be programmed to do this.

**File Manipulation** – this involves reading and writing files, creating and deleting files and folders. It also involves searching and listing files by name. It also involves granting and denying access to files and folders based upon ownership.

# Operating-System Services

The command **Attrib** displays, sets, or removes the read-only, archive, system, and hidden attributes assigned to files or directories. Used without parameters, **attrib** displays attributes of all files in the current directory.

```
attrib [{+r|-r}] [{+a|-a}] [{+s|-s}] [{+h|-h}] [[Drive:][Path] FileName] [/s[/d]]
```

## Parameters

**+r** : Sets the read-only file attribute.

**-r** : Clears the read-only file attribute.

**+a** : Sets the archive file attribute.

**-a** : Clears the archive file attribute.

**+s** : Sets the system file attribute.

**-s** : Clears the system file attribute.

**+h** : Sets the hidden file attribute.

**-h** : Clears the hidden file attribute.

# Operating-System Services

The command **chmod** changes the permission of a file.

Syntax

- `chmod [OPTION]... MODE[,MODE]... FILE...`  
`chmod [OPTION]... OCTAL-MODE FILE... Numeric Permissions:`

## CHMOD Numeric Permissions:

400 read by owner  
040 read by group  
004 read by anybody (other)  
200 write by owner  
020 write by group  
002 write by anybody  
100 execute by owner  
010 execute by group  
001 execute by anybody

## Permissions

*u* – Owner/User who owns the file.  
*g* - Group that owns the file.  
*o* - Other.  
*a* - All.  
*r* - Read the file.  
*w* - Write or edit the file.  
*x* - Execute or run the file as a program.

The command **chmod 755 file.txt** would yield the permission

-	rwx	r-x	r-x	1	hope	123	Feb 03 15:36	file.txt
File	Owner	group	everyone else	links	owner	size	mod date	file name

# Operating-System Services

**Communications** – being that computers are often times communicating with other computers normally through a network, where memory is shared, or through message passing, information is moved between processes by the operating system in packets (formatted units of data).

**Error Detection** – The operating system needs to be constantly aware of possible errors. Errors may occur in the CPU and memory hardware (such as a memory error or a power failure), in I/O devices (such as a parity error on tape, a connection failure on a network, or lack of paper in the printer), and in the user program (such as an arithmetic overflow, an attempt to access an illegal memory location, or a too-great use of CPU time). For each type of error, the operating system should take the appropriate action to ensure correct and consistent computing.



# Operating-System Services

Resource Allocation, Accounting and Protection and Security are operating system functions that exist for the purpose of ensuring the efficient operation of the system.

**Resource allocation** - it becomes important when there are multiple users or multiple jobs running at the same time; resources must be allocated to each of them. (i.e. CPU cycles, main memory and file storage)

**Accounting** – It may come a time when we may want to keep track of each users time usage of resources. This may be done for billing purposes or simply to accumulate statistics on the usage of resources in order to improve your computing service.

**Protection and Security** – Protection involves ensuring that all access to system resources are controlled (i.e. process interference). Security through passwords authenticate each user attempting to gain access to system resources.

# User Operating-System Interface

As mentioned earlier there are several ways for a user to interface with the operating system.

1. Command line
2. Batch processing
3. Graphical User Interfaces (Icons)

## **Command Interpreter**

Some operating systems include the command interpreter in the kernel. Others, such as Windows and UNIX, treats the command interpreter as a special program that is running when a job is initiated or when a user first logs on.

On system with multiple command interpreters to choose from, the interpreters are know as shells.

## UNIX and LINUX Shells

(Korn, Bourne, C and Bourne-Again)

# User Operating-System Interface

## **Command Interpreter (cont.)**

The main function of the command interpreter is to get and execute the next user-specified command. Many of the commands given at this level manipulate files: create, delete, list, print, copy, execute, etc.

## **Graphical User Interfaces**

A second strategy for interfacing with the operating system is through a user friendly graphical user interface, or GUI. Rather than entering commands directly via a command-line interface, users employ a mouse-based window-and-menu system on a computer's desktop. Depending on the mouse pointer's location, clicking a mouse button will invoke a program, select a file or directory (folder), or pull down a menu that contains commands.

# User Operating-System Interface

## **Graphical User Interfaces (cont.)**

GUI's first appeared due in part to research taking place in the early 1970s at Xerox PARC research facility. The first GUI appeared on the Xerox Alto computer in 1973 and soon became more widespread with the advent of Apple Macintosh computers in the 1980s. Microsoft's first version of Windows, (Windows – Version 1.0), was based on the addition of a GUI interface to the MS-DOS operating system.

# User Operating-System Interface

## **System Calls**

System calls provide an interface to the services made available by an operating system. These calls are generally available as routines written in C and C++, although certain low-level tasks may need to be written when hardware must be accessed directly.

System calls are invoked when instructions, are displayed, a file needs to read an input file, write to an output file, close both files and writing a confirmation message.

# User Operating-System Interface

Example of how system calls are used.

source file

destination file

Write a program using these stages of system calls to simulate copying a data file.

Example System Call Sequence

- Acquire input file name
  - Write prompt to screen
  - Accept input
- Acquire Output file name
  - Write prompt to screen
  - Accept input
- Open the input file
  - if file doesn't exist, abort
- Create output file
  - if file exists abort
- Loop
  - Read from input file
  - Write to output file
- Until read fails
- Close output file
- Write completion message to screen
- Terminate normally

# User Operating-System Interface

## **Types of System Calls**

1. Process control
2. File manipulation
3. Device manipulation
4. Information manipulation
5. Communications
6. Protection

# User Operating-System Interface

## Types of System Calls

1. Process control – the halting of a running program's execution whether it halts normally (programs end) or abnormally (aborted).

If a system call is made to terminate the currently running program abnormally, or if the program runs into a problem and causes an error trap, a dump of memory is sometimes taken and an error message generated. The dump is written to disk and may be examined by a debugger (system programmer).

Under either normal or abnormal circumstances, the operating system must transfer control to the invoking command interpreter.



# User Operating-System Interface

## Types of System Calls (Process Control )

**Interactive system**, the command interpreter simply continues with the next command; it assumes that the user will issue an appropriate command to respond to an error.

**GUI system**, a pop-up window might alert the user to the error and ask for guidance.

**Batch system**, the command interpreter usually terminates the entire job and continues with the next job.

# User Operating-System Interface

## Types of System Calls (Process Control )

### Summary of System calls

- 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

Example of Windows and Unix System Calls		
	Windows	Unix
Process Control	CreateProcess()	fork()
	ExitProcess()	exit()
	WaitForSingleObject()	wait()

# User Operating-System Interface

## Types of System Calls

2. File Management– the creation, deleting, reading, writing or repositioning of files.

These system calls require the name of the file and perhaps some of the file's attributes. After the action of the file has been invoked, we need to open it for use. Once the action of the file has been completed, we need to close the file, indicating that we no longer want to use it.

We may need these same sets of operations for directories if we have a directory structure for organizing files in the file system. In addition, for either files or directories, we need to be able to determine the value of various attributes and perhaps to reset them if necessary.

# User Operating-System Interface

## Types of System Calls (File Management)

### File Attributes

- File size in bytes
- file date/time (*creation time, last-modify time, last-access time*)
- file attributes archive bit (*shows that the file has not been archived yet*)
- read-only bit (*write-protect the file*) directory bit (*distinguishes a directory from a file*)
- hidden bit (*hides from an ordinary directory listing*)
- system bit (*denotes a system file whatever that means*)

# User Operating-System Interface

## Types of System Calls (File Management)

### Summary of System calls

- Create file, Delete file
- Open, Close
- Read, Write, Reposition
- Get File Attributes, Set File Attributes

Example of Windows and Unix System Calls		
	Windows	Unix
File Management	CreateFile()	open()
	ReadFile()	read()
	WriteFile()	write()
	CloseHandle()	close()

# User Operating-System Interface

## Types of System Calls

3. Device Management– making resources available for process execution.

A process may need several resources to execute (i.e. main memory, disk drives, access files, etc.). If resources are available, they can be granted and control can be returned to the user process. Otherwise, the process will have to wait until sufficient resources are available. Resources can also be thought of as devices (physical – disk drives, abstract - files).

A system with multiple users may require us to first request the device, to ensure exclusive use of it. Then, once finished with the device, release it. (Similar to file management)

So similar that many OS merge the two into a combined file-device structure.

# User Operating-System Interface

## Types of System Calls (Device Management)

### Summary of System calls

- Request device, release device
- Read, Write, reposition
- Get Device Attributes, Set Device Attributes
- Logically attach or detach devices

Example of Windows and Unix System Calls		
	Windows	Unix
Device Management	SetConsoleMode()	ioctl()
	ReadConsole()	read()
	WriteConsole()	write()

# User Operating-System Interface

## **Types of System Calls** (Information Maintenance)

4. Information Maintenance – transferring information between the user program and the operating system.
  - Program trace – lists each system call as it is executed
  - Time profile – indicate the amount of time a program executes at a specific location or set of locations

The operating system keeps information about all its processes.



# User Operating-System Interface

## Types of System Calls (Information Maintenance)

### Summary of System calls

- Get time or date, Set time or Date
- Get system date, Set system date
- Get process, file or device attribute
- Set process, file, or device attribute

Example of Windows and Unix System Calls		
	Windows	Unix
Information Maintenance	GetCurrentProcessID()	getpid()
	SetTimer()	alarm()
	Sleep()	sleep()

# User Operating-System Interface

## Types of System Calls

5. Communication – (two common models)
  - a) **Message-passing model** – communicating processes exchange messages with one another to transfer information.
  - a) **Shared-memory model** - processes used *shared memory create* and *shared memory attach* system calls to create and gain access to regions of memory owned by other processes

# User Operating-System Interface

## Types of System Calls (Communication)

- a) **Message-passing model** - Messages can be exchanged between processes either directly or indirectly through a common mailbox
1. A connection must be opened
  2. The name of other communicator must be known. Can be;
    1. Another process on the same system
    2. A process on another computer connected by a communications network (i.e. host name, ip address, process name), which is translated into an identifier in which the operating system refers to.

Both of the models are common in operating systems, and most systems implement both. Message passing is useful for exchanging smaller amounts of data, because no conflicts need be avoided. It is also easier to implement than is shared memory for intercomputer communication. Shared memory allows maximum speed and convenience of communication, since it can be done at memory transfer speed.

# User Operating-System Interface

## Types of System Calls (Communication)

- a) **Shared-memory model** – requires that two or more processes agree to remove the restriction of preventing one process from accessing another process's memory.
  1. Information is exchanged by reading and writing data in shared areas.
  2. The form of data is determined by the process (is not under the OS control)
  3. Process are responsible for ensuring that they are not writing to the same memory location simultaneously

The **get hostid** and **get processid** system calls do this translation. The identifiers are then passed to the general-purpose open and close calls provided by the file system or to specific open connection and close connection system calls. The recipient process must usually give its permission for communication to take place with an **accept connection** call.

# User Operating-System Interface

## Types of System Calls (Communication)

### Summary of System calls

- Create, delete communication connection
- Send, receive messages
- Transfer status information
- Attach or detach remote devices.

Example of Windows and Unix System Calls		
	Windows	Unix
Information Maintenance	CreatePipe()	pipe()
	CreateFileMapping()	shmget()
	MapViewOfFile()	mmap()

# User Operating-System Interface

## Types of System Calls

4. Protection – provides the mechanism for controlling access to the resources provided by a computer system

All computer systems, from servers to PDA's, must be concerned with protection. Typically, system calls providing protection include **set permission** and **get permission**, which manipulate the permission settings of the resources such as files and disk. **The allow user** and **deny user** system calls specify whether particular users can – or cannot – be allowed access to certain resources.

# User Operating-System Interface

**System Programs** – also known as system utilities, provides a convenient environment for program development and execution. Some of them are simply interfaces to system calls; others are considerably more complex. These are the categories:

- **File management** – which are programs that create, delete, copy, rename, print, dump, list, and generally manipulates files and directories.
- **Status information** – the collecting of data pertaining to the system or process (i.e. system date, available memory, disk space, number of users, other status information, performance details, and logging, and debugging information.
- **File modification (text editors)** – the ability to create and modify the content of files stored on disk or other storage devices; may even include searches or transformation of text.

# User Operating-System Interface

- **Programming-language support** – Compilers, assemblers, debuggers, and interpreters for common programming languages (such as C, C++, Java, Visual Basic, and PERL) are often provided to the user with the operating system.
- **Program loading and execution** – Once a program is assembled or compiled, it must be loaded into memory to be executed. The system may provide absolute loaders, relocatable loaders, linkage editors, and overlay loaders. Debugging systems for either high-level languages or machine languages are needed as well.
- **Communications** – These programs provide the mechanism for creating virtual connections among processes users, and computer systems. They allow users to send messages to one another's screens, to browse Web pages, to send electronic-mail, to log in remotely, or to transfer file from one machine to another.



# Operating-System Structure

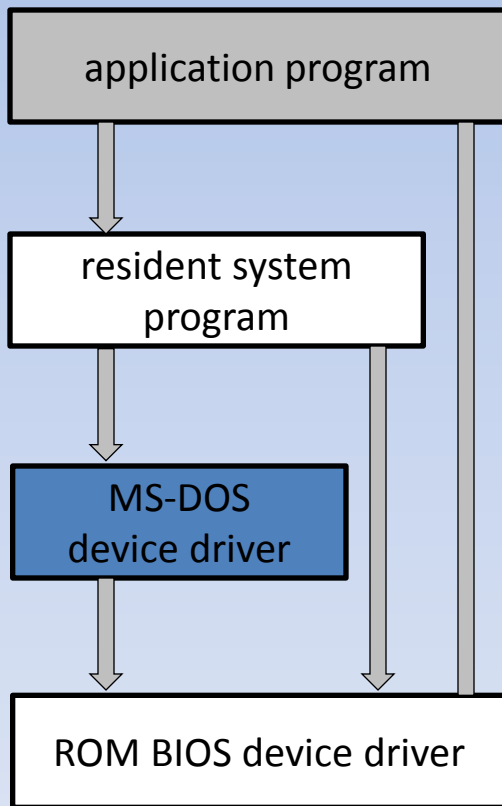
- Simple Structure

Many commercial operating systems do not have well-defined structures. Frequently, such systems started as small, simple, and limited systems and then grew beyond their original scope. MS-DOS is an example of such a system. It was originally designed for a few people who had no idea that it would become so popular. It was written to provide the most functionality in the least space, so it was not divided into modules carefully.

Another example would be the original UNIX operating system. Like MS-DOS, UNIX initially was limited by hardware functionality. It consists of two parts: the kernel and the system programs.

# Operating-System Structure

## MS-DOS Layer Structure



In MS-DOS, application programs are able to access the basic I/O routines to write directly to the display and disk drives. Such freedom leaves MS-DOS vulnerable to errant (or malicious) programs, causing entire system crashes when user programs fails.

# Operating-System Structure

- Layered Approach

With proper hardware support, operating systems can be broken into pieces that are smaller and more appropriate than those allowed by the original MS-DOS and UNIX systems.

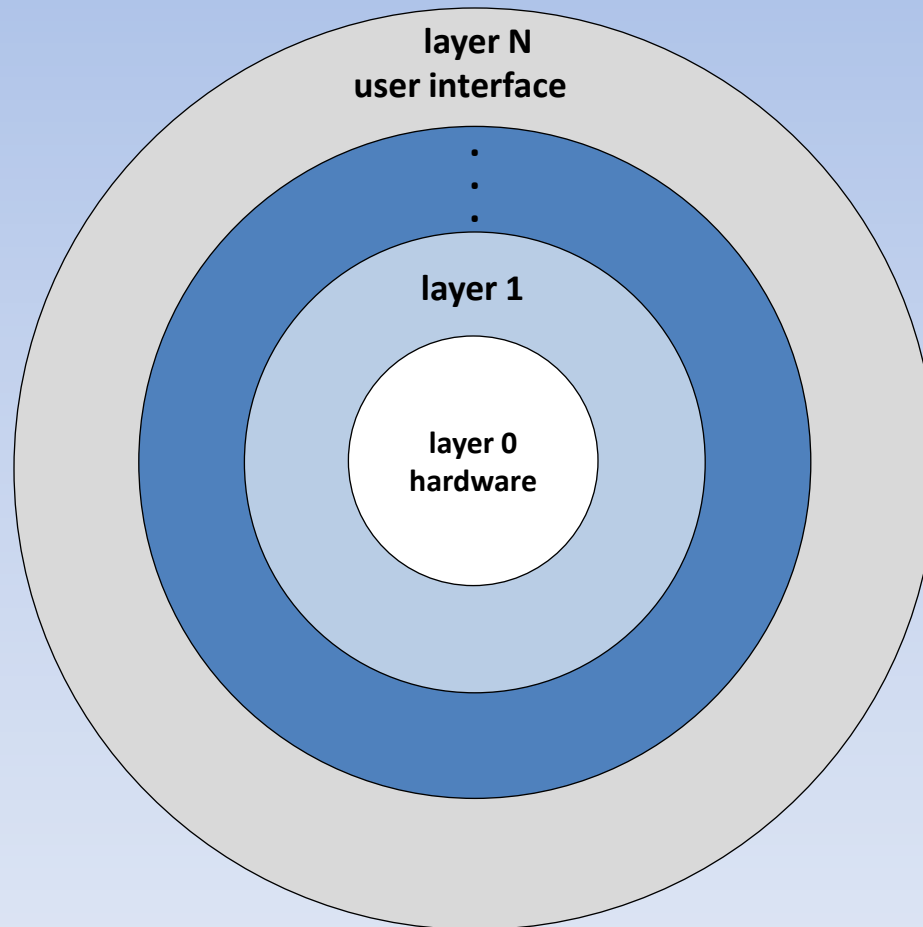
A system can be made modular in many ways.

1. Layered Approach (levels) – where the bottom layer (layer 0) is hardware and the highest layer (layer N) is the user interface.

An operating-system layer is an implementation of an abstract object made up of data and the operations that can manipulate that data. A typical operating system layer – say, layer M – consists of data structures and a set of routines that can be invoked by higher-level layers. Layer M, in turn can invoke operations on lower-level layers.

# Operating-System Structure

- Layered Approach



# Operating-System Structure

- Layered Approach

The main advantage of the layered approach is simplicity of construction and debugging. The first level can be debugged without concern for the rest of the system, because, by definition, it uses only the basic hardware (which we assume is correct) to implement its functions. Once the first layer is debugged, its correct functioning can be assumed while the second layer is debugged, and so on.

The layered approach is simplified because each layer is implemented with only those operations provided by lower-level layers. A layer does not need to know how these operations are implemented; it needs to know only what these operations do.

# Operating-System Structure

- Microkernels

Mach OS developed in mid-1980s at Carnegie Mellon University modularized the UNIX kernel using the microkernel approach.

This method structures the operating by removing all nonessential components from the kernel and implementing them as system and user-level programs, which results to a smaller kernel. It provides minimal process and memory management, in addition to a communication facility.

The main function of the microkernel is to provide a communication facility between the client program and the various services that are also running in user space, which provided by message passing.

Microkernels can suffer from performance decreases due to increased system functions (i.e. Window NT).

# Operating-System Structure

- Modules

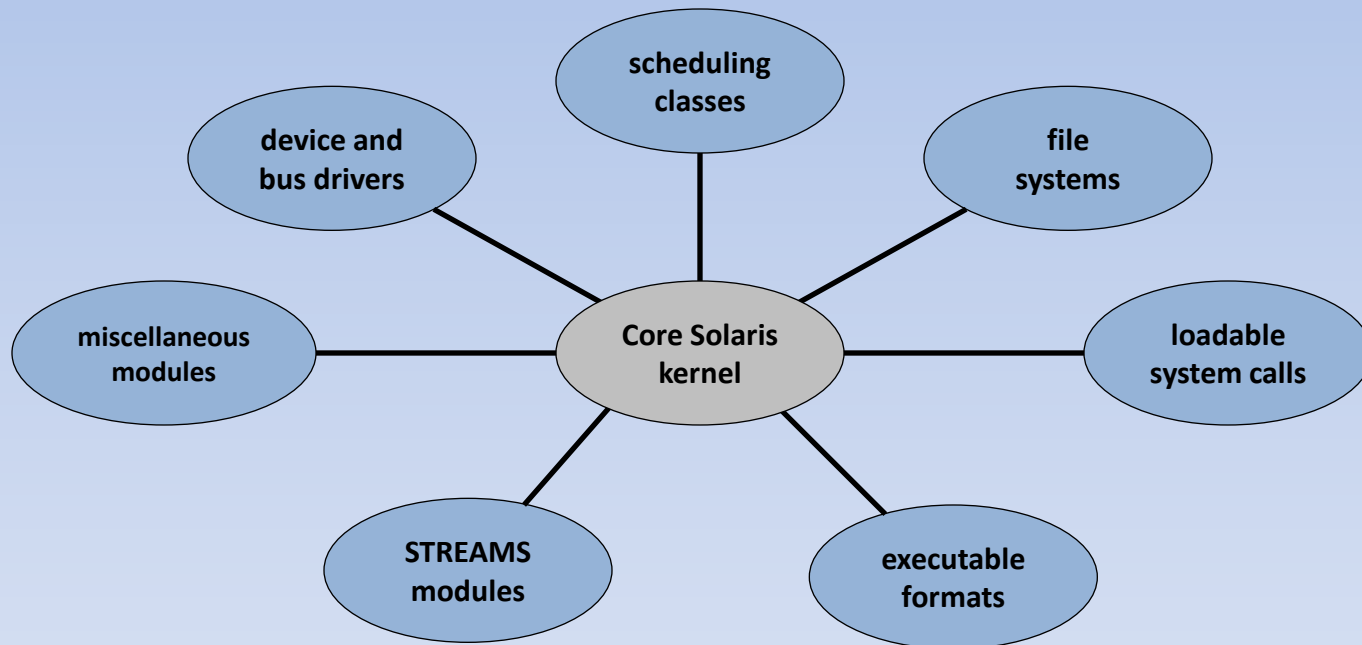
Considered the best current methodology for operating-system design because it involves using object-oriented programming techniques to create a modular kernel.

The kernel has a set of core components and links in additional services either during boot time or during run time. This strategy uses dynamically loadable modules and is common in modern implementations of UNIX, such as Solaris, Linux, and Mac OS X.

This design allows the kernel to provide core services yet also allows certain features to be implemented dynamically. For example, device and bus drivers for specific hardware can be added to the kernel, and support for different file systems can be added as loadable modules.

# Operating-System Structure

- Solaris Loadable Modules





# Operating-System Structure

- Modules

The overall result resembles a layered system in that each kernel section has defined, protected interfaces; but it is more flexible than a layered system in that any module can call any other module. The approach is like the microkernel approach in that the primary module has only core functions and knowledge of how to load and communicate with other modules; but it is more efficient, because modules do not need to invoke message passing in order to communicate.