Module 1

Introduction to OS

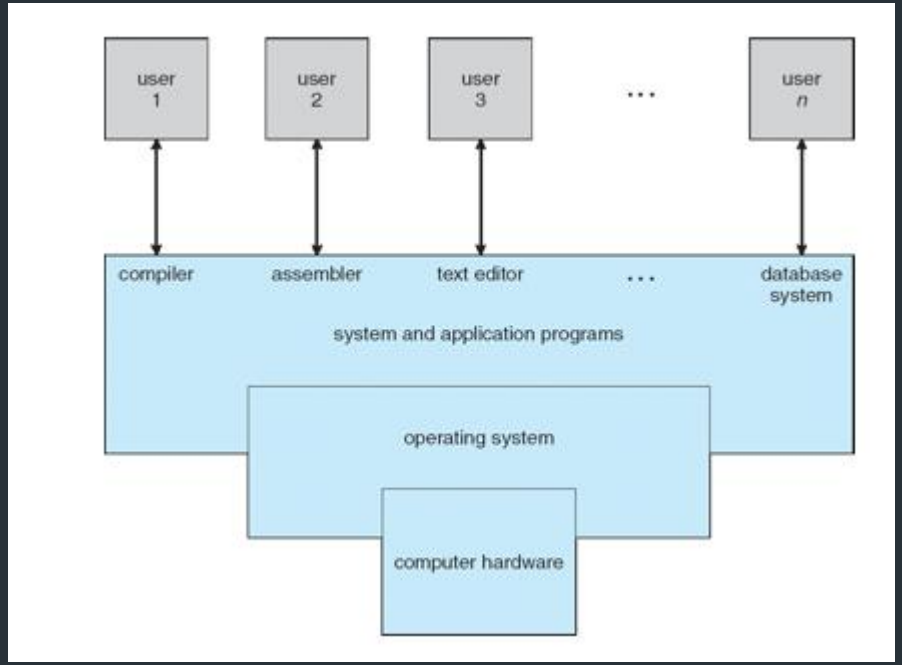
VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

What is an Operating System?

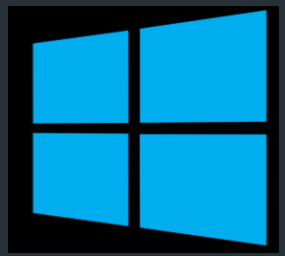
- A program that acts as an intermediary between a user of a computer and the computer hardware
- Operating system goals:
  - Execute user programs and make solving user problems easier
  - Make the computer system convenient to use
  - Use the computer hardware in an efficient manner



Source: "Operating System Concepts: 9th Edition" by Silberschatz Galvin Gagne

Introduction to OS

- Operating System(OS) manages computer hardware.
- Application Program runs on OS
- Computer User interacts with the OS which in turn interacts with the hardware.
- Variety of OS depending on the tasks. Example:-
  - Mainframe OS
    - Optimize hardware utilization
  - Computer standard OS
    - Standard Application, Games, etc.
  - Handheld OS -> Apps, etc.



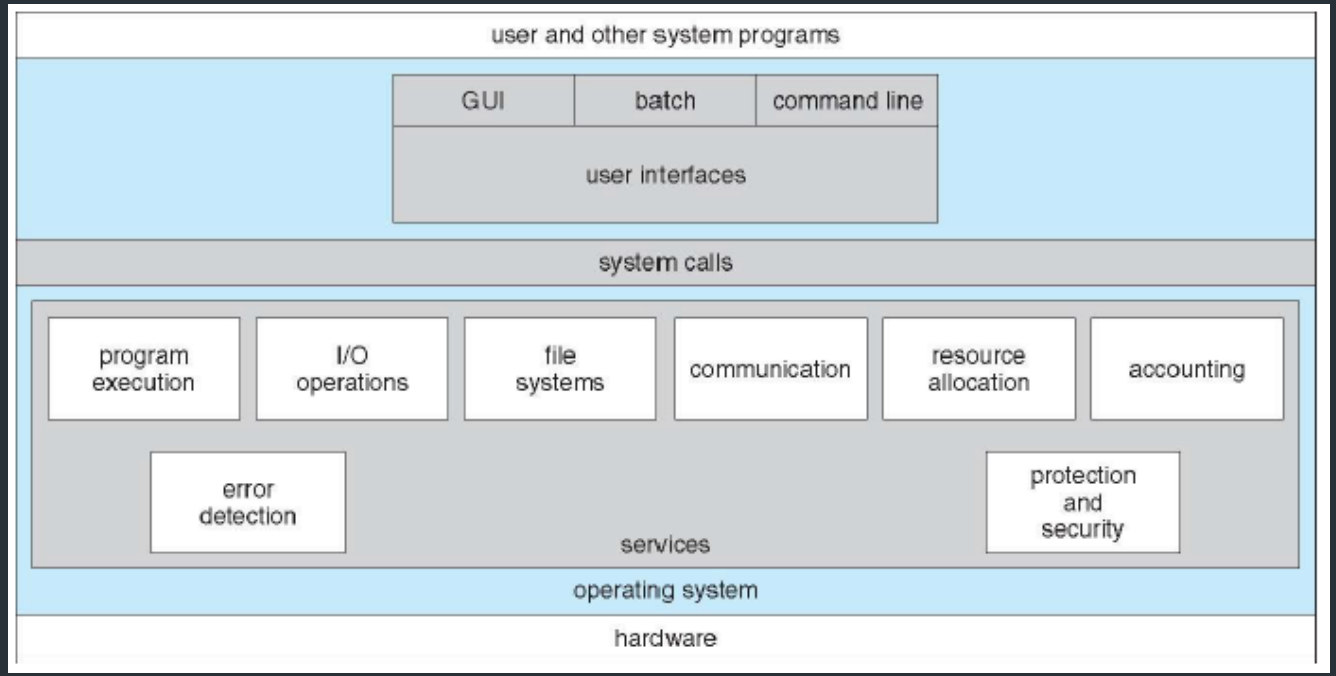

iOS

OS as a base for Application Programs

OS is a resource allocator
Manages all resources
Decides between conflicting requests for efficient and fair resource use
OS is a control program
Controls execution of programs to prevent errors and improper use of the computer

- Application program runs on a platform and that platform is an Operating systems.
- OS plays an important role to determine which application you need, because some applications may exists only in some OS.
- Example:
  - Words in windows
  - Libre office in linux

Schematic of Operating System Operations



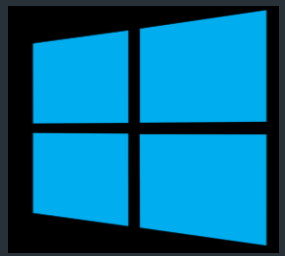
Silberschatz, Gagne, Galvin: Operating System Concepts, 6th Edition

OS Services

- **User Interface:** There are different kinds, like touchscreen, GUI, and command-line.
- **Program Execution:** (Execute programs for users)
- **I/O operations:** It is much too difficult for users to operate the I/O hardware correctly without help.
- **File System Manipulation:** The OS helps us store, organize, manage, and protect our information.
- **Communications:** Users need their processes to exchange information. OSs help. The two main ways to do it are *with shared memory* and *by message passing*.
- **Error Detection:** An OS continually checks to see if something is going wrong. The OS is programmed to take appropriate action.
- **Resource Allocation**
- **Logging:**
  - Records for accounting, fault detection, failure, protection, maintenance, update, security, etc.
- **Protection and Security**.

Different types of OS

- Microsoft Windows
- Mainframe
- DOS
- OS/2
- Linux - Example Ubuntu
- Mac OS
- AmigaOS



Design Goals

- **Design Goals:**
  - system that is convenient,
  - reliable,
  - safe, and
  - fast.
- **Implementation:** **The** *implementation* of the operating system, that is the manner in which the ideas of the design are written in programming language(s).
  - Assembly
  - High Level Language
- Earlier assembly could make the code run faster but nowadays high-level are translated to equivalently good assembly code.
- Instead performance of OS will increase if selection data structure and algorithms are done rather than proper assembly code.

OS Design Issues

- Efficiency – amount of useful work wrt the time and resources
- Robustness – prolonged working hours without crashing
- Flexibility – ease of modification of components
- Portability – ability to run a program in different platforms
- Security – secure systems – authentication and authorization
- Compatibility – software runs on different models of the family.

Operating System Structures

- Simple
- Monolithic
- Micro-kernel models
- Layered (conceptual)
- Modular

Kernels may be classified mainly in three categories: -

Monolithic Kernel
Micro Kernel
Hybrid Kernel

Simple Structure

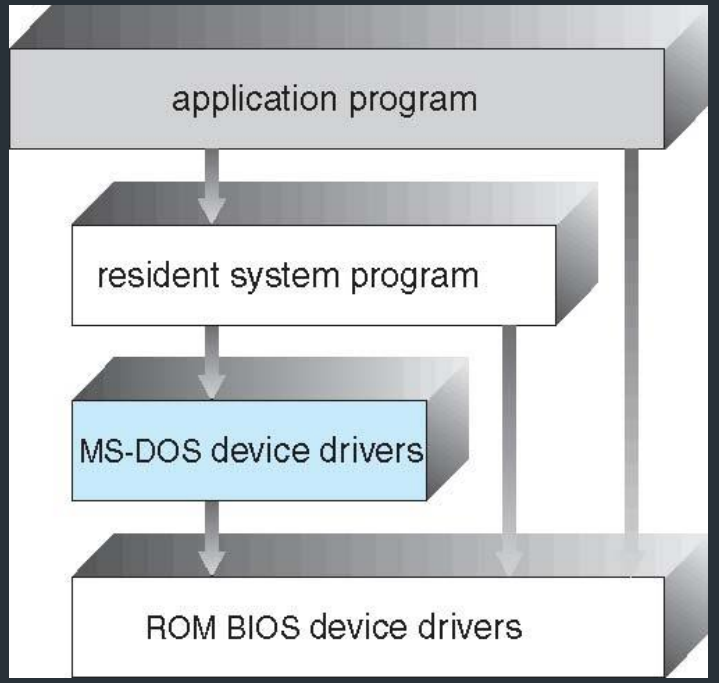
Advantages of Simple structure:

- It delivers better application performance because of the few interfaces between the application program and the hardware.
- Easy for kernel developers to develop such an operating system.

Disadvantages of Simple structure:

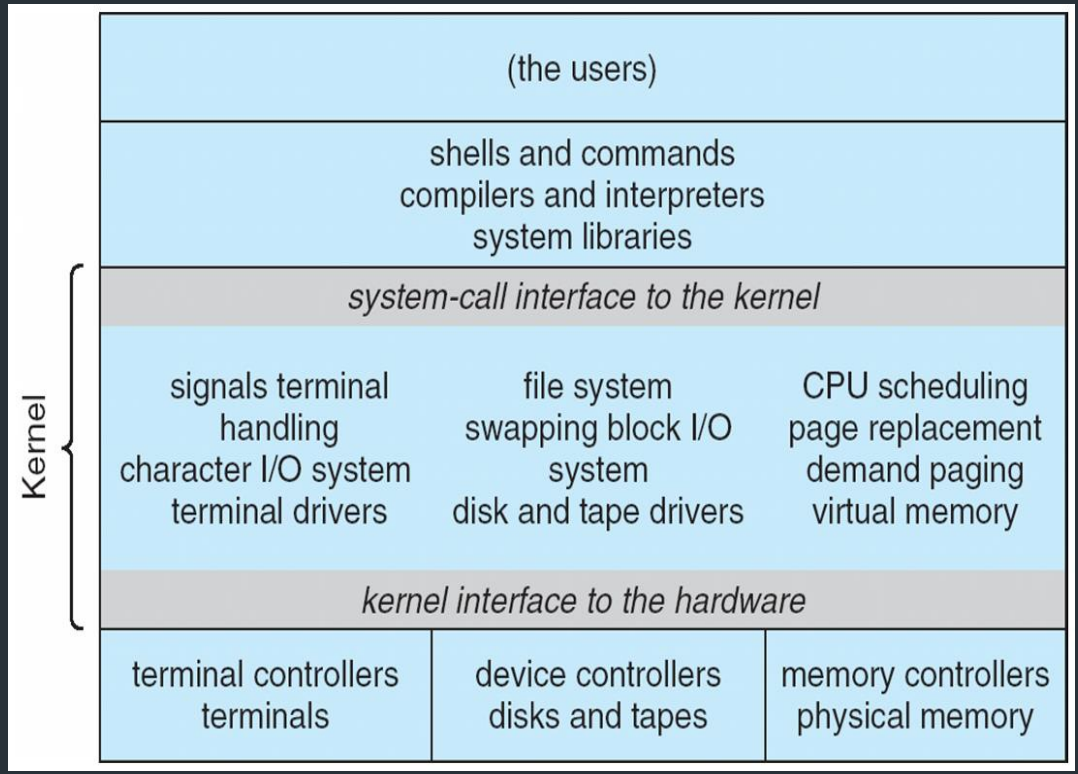
- The structure is very complicated as no clear boundaries exists between modules.
- It does not enforce data hiding in the operating system.

- Ex – MS DOS



Monolithic Structure

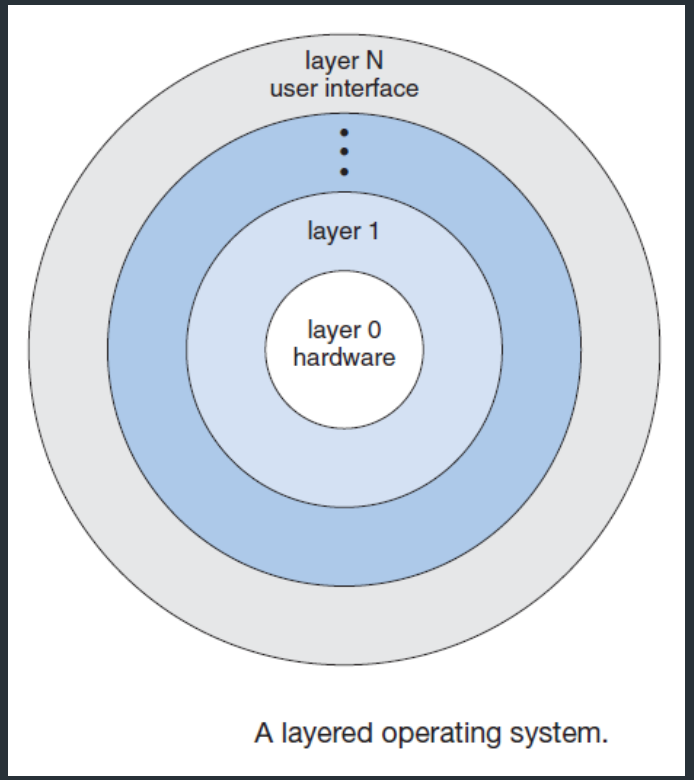
- Monolithic Kernel
  - value on speed and efficiency.
  - Monolithic is a single static binary file.
  - It executes in a single address space.
  - Debugging and modification is difficult
  - Too many things packed in one level
  - Ex – Earlier Unix



Silberschatz, Gagne, Galvin: Operating System Concepts, 6th Edition

Layered Structure

- Division into number of layers as shown in figure.
- Innermost layer is hardware
- Outermost layer is interface
- Advantages of Layered structure:
  - ✓ Layering makes it easier to enhance the operating system as implementation of a layer can be changed easily without affecting the other layers.
  - ✓ It is very easy to perform debugging and system verification.
  - ✓ Hardware protection is provided by the layers above.
- Disadvantages of Layered structure:
  - ✓ In this structure, the application performance is degraded as a simple request might need to pass through many layers below it.
  - ✓ It requires careful planning for designing the layers as higher layers use the functionalities of only the lower layers.

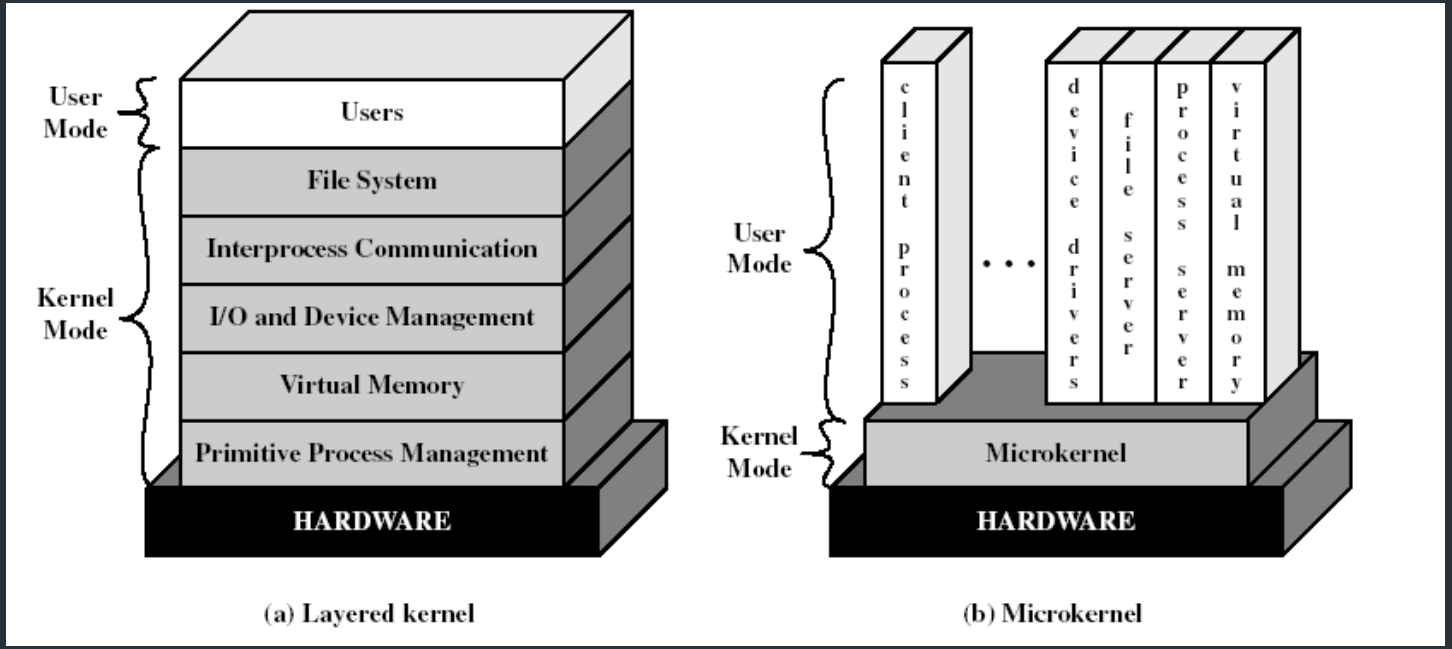


A layered operating system.

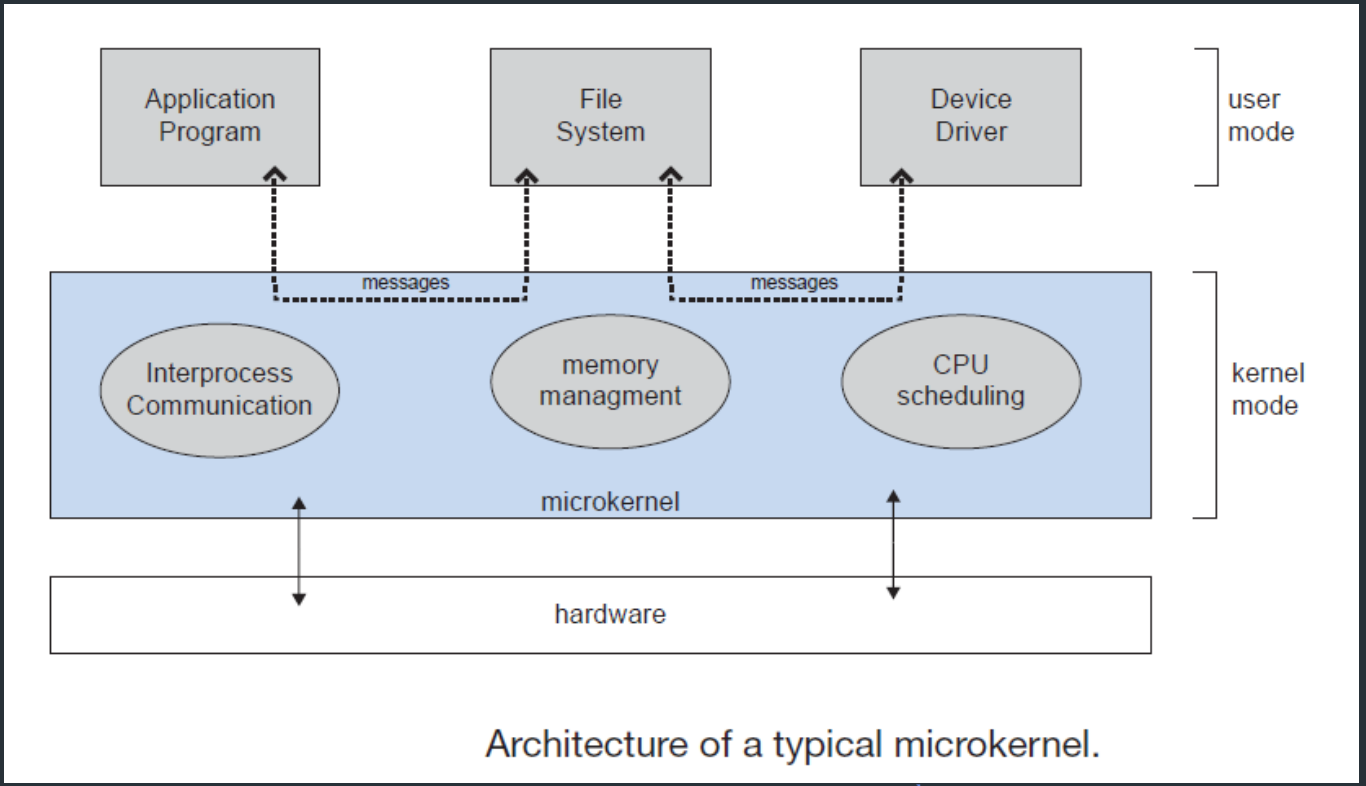
Silberschatz, Gagne, Galvin: Operating System Concepts, 6th Edition

Microkernel Structure

- Keep only necessary component in kernel. Others are implemented as programs (system or user level).
- Resulting in a kernel smaller in size.
- Minimal process management
- Minimal memory management
- Main role is that it facilitates communication between the client program and the various services that are running in user space.
- **Mach** example of **microkernel**
  - Mac OS X kernel (**Darwin**) partly based on Mach
- 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 (less code is running in kernel mode)
  - More secure
- Detriments:
  - Performance overhead of user space to kernel space communication


User Mode
Kernel Mode
Users
File System
Interprocess Communication
I/O and Device Management
Virtual Memory
Primitive Process Management
HARDWARE
(a) Layered kernel
User Mode
Kernel Mode
client process
device drivers
file server
process server
virtual memory
Microkernel
HARDWARE
(b) Microkernel

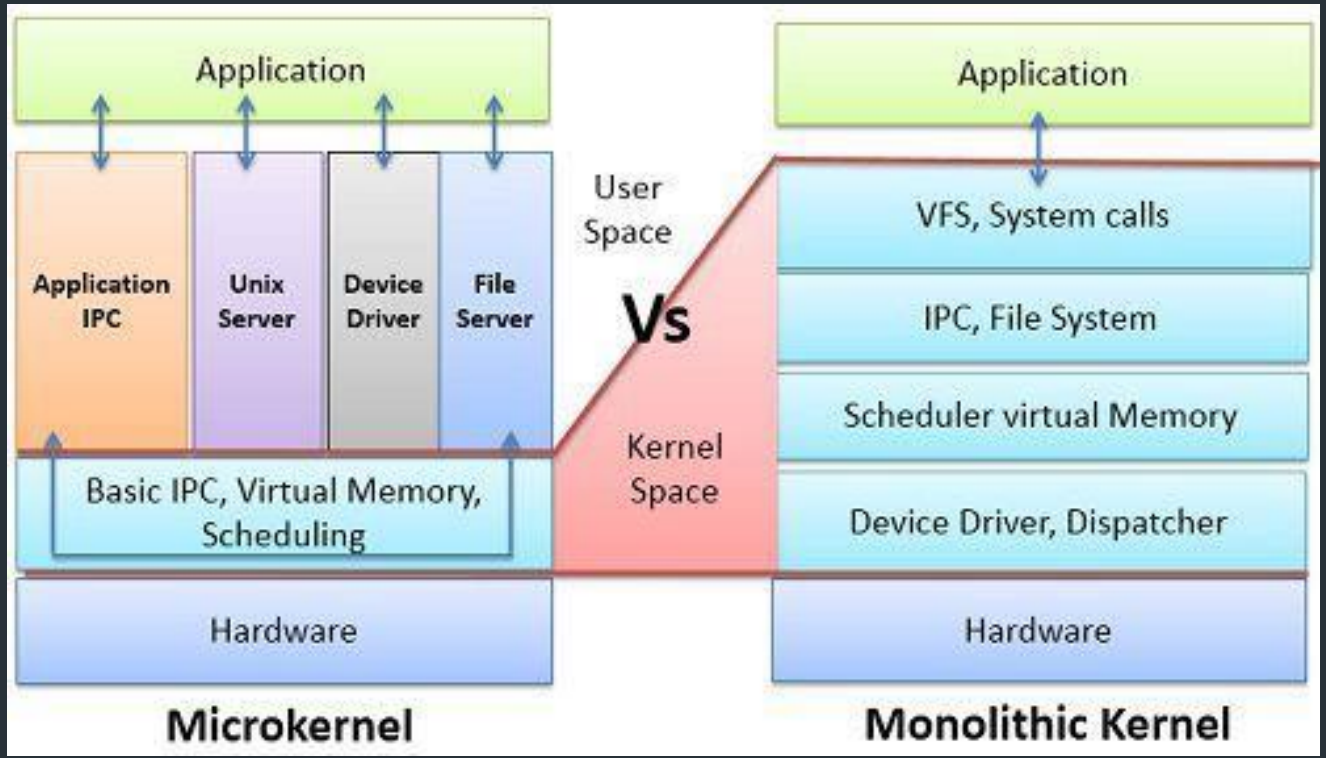
Microkernel



Architecture of a typical microkernel.

Silberschatz, Gagne, Galvin: Operating System Concepts, 6th Edition

Microkernel vs Monolithic

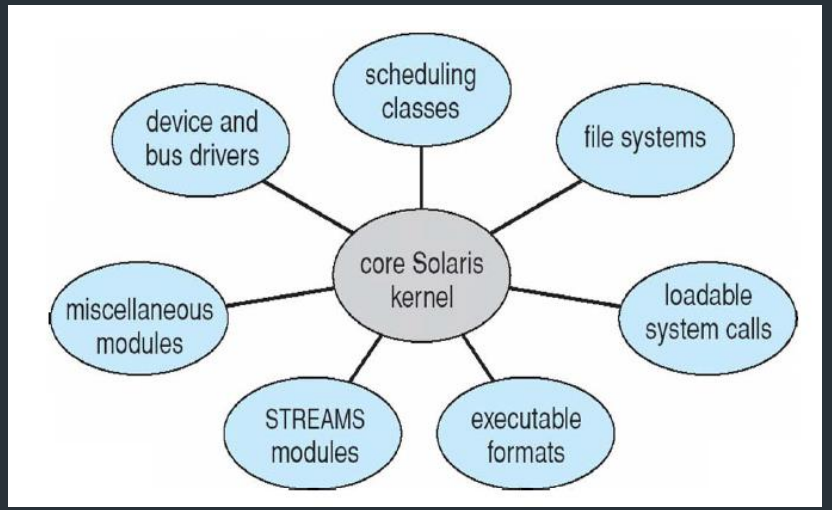


Source: www.atg.world/Monolithic vs Microkernel

BASIS FOR COMPARISON	MICROKERNEL	MONOLITHIC KERNEL
Basic	In microkernel user services and kernel, services are kept in separate address space.	In monolithic kernel, both user services and kernel services are kept in the same address space.
Size	Microkernel are smaller in size.	Monolithic kernel is larger than microkernel.
Execution	Slow execution.	Fast execution.
Extendible	The microkernel is easily extendible.	The monolithic kernel is hard to extend.
Security	If a service crashes, it does effect on working of microkernel.	If a service crashes, the whole system crashes in monolithic kernel.
Code	To write a microkernel, more code is required.	To write a monolithic kernel, less code is required.
Example	QNX, Symbian, L4Linux, Singularity, K42, Mac OS X, Integrity, PikeOS, HURD, Minix, and Coyotos.	Linux, BSDs (FreeBSD, OpenBSD, NetBSD), Microsoft Windows (95,98,Me), Solaris, OS-9, AIX, HP-UX, DOS, OpenVMS, XTS-400 etc.

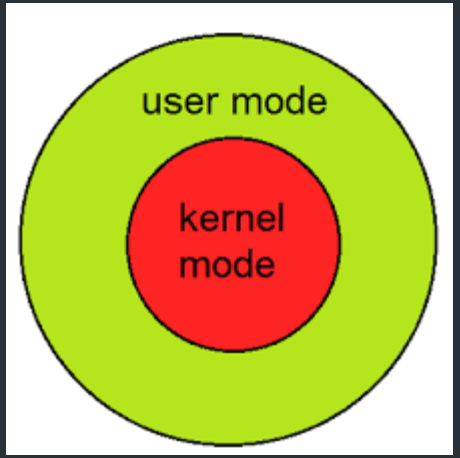
Modular

- Divided into different module.
- Typically employs:
  - dynamic loadable kernel module (LKM). i.e. Different modules communicate through kernel (core part).
- LKM may be loaded during boot or when required, and can be deleted also.
- An example would be a device driver support module loaded when a new device is plugged into the computer, and when the device is unplugged, the module is deleted because it is not needed any more.



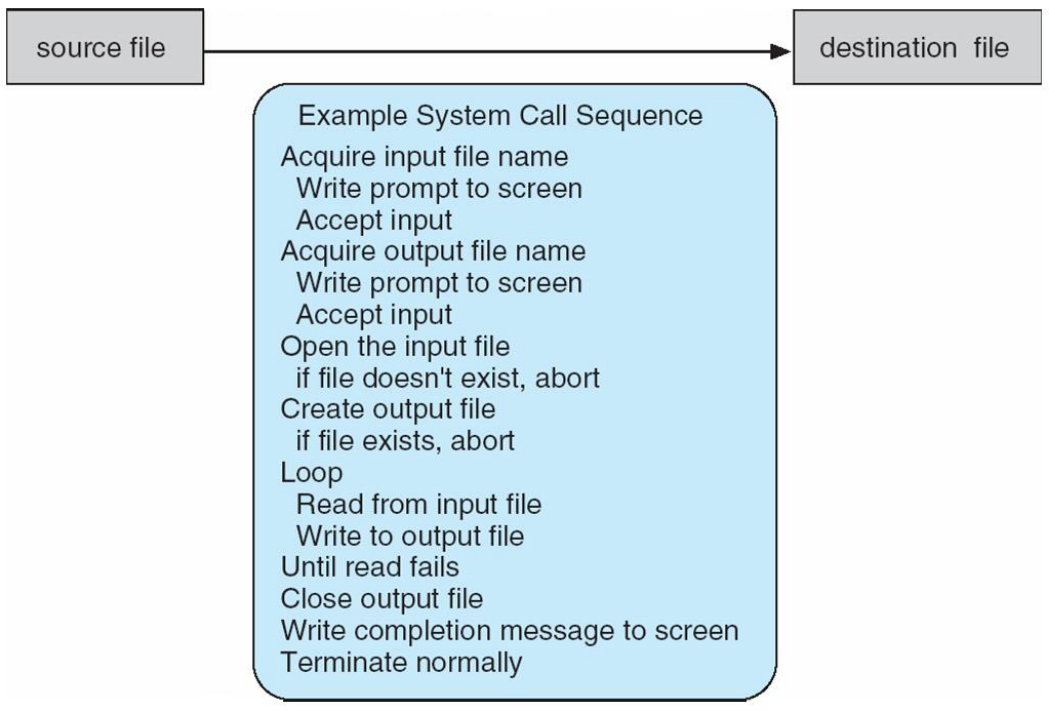
System Calls

- Programming interface to the services provided by the OS
- Typically written in a high-level language (C or C++)
- It is a programmatic way in which a computer program requests a service from the kernel of the OS
- 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 (including virtually all versions of UNIX, Linux, and Mac OS X), and Java API for the Java virtual machine (JVM)

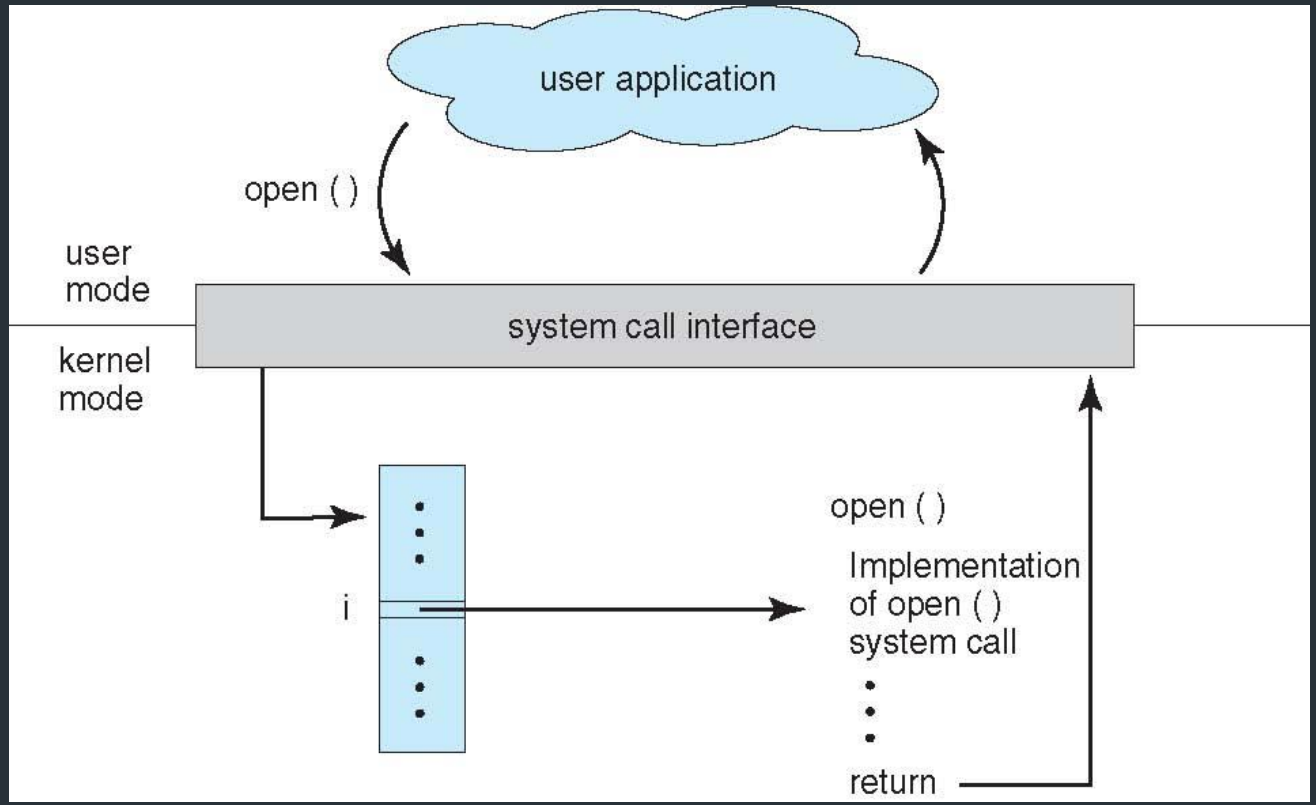


System Calls - Example

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



API – System Call – OS



Example of a Standard API

EXAMPLE OF STANDARD API

As an example of a standard API, consider the `read()` function that is available in UNIX and Linux systems. The API for this function is obtained from the `man` page by invoking the command

```
man read
```

on the command line. A description of this API appears below:

```
#include <unistd.h>

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

return value — function name — parameters

A program that uses the `read()` function must include the `unistd.h` header file, as this file defines the `ssize_t` and `size_t` data types (among other things). The parameters passed to `read()` are as follows:

- `int fd`—the file descriptor to be read
- `void *buf`—a buffer where the data will be read into
- `size_t count`—the maximum number of bytes to be read into the buffer

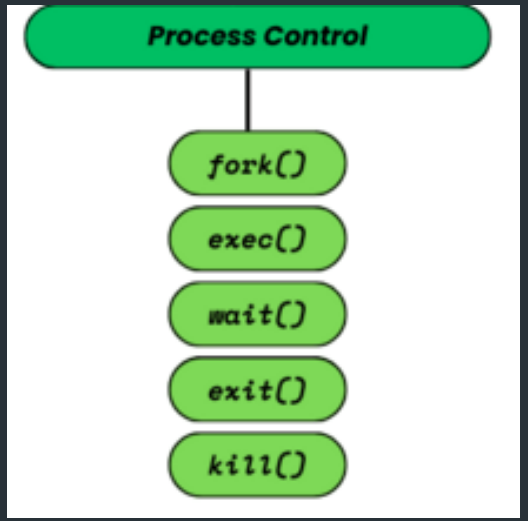
On a successful read, the number of bytes read is returned. A return value of 0 indicates end of file. If an error occurs, `read()` returns −1.

System Call – Types

- Process control
- File manipulation
- Device Management
- Information Maintenance
- Communications

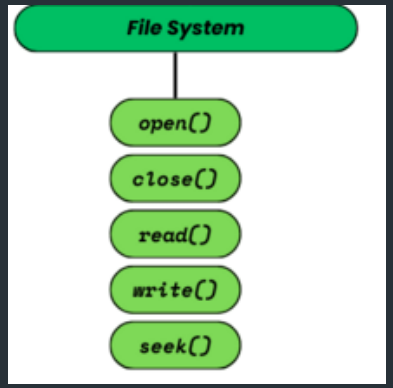
System Call – Types

- 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

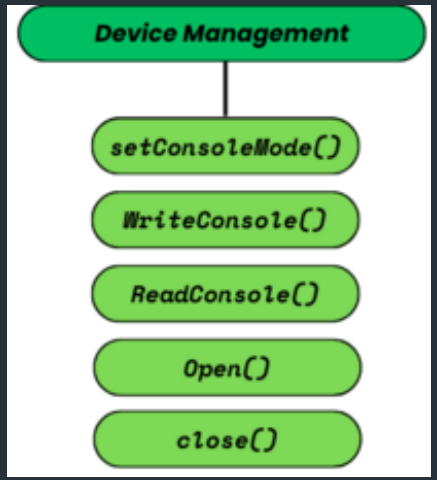


System Call – Types (Cont.)

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

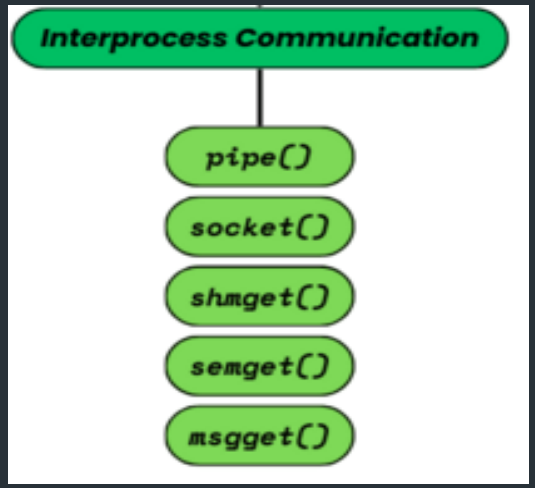


- Device management
  - request device, release device
  - read, write, reposition
  - get device attributes, set device attributes
  - logically attach or detach devices



System Call – Types (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

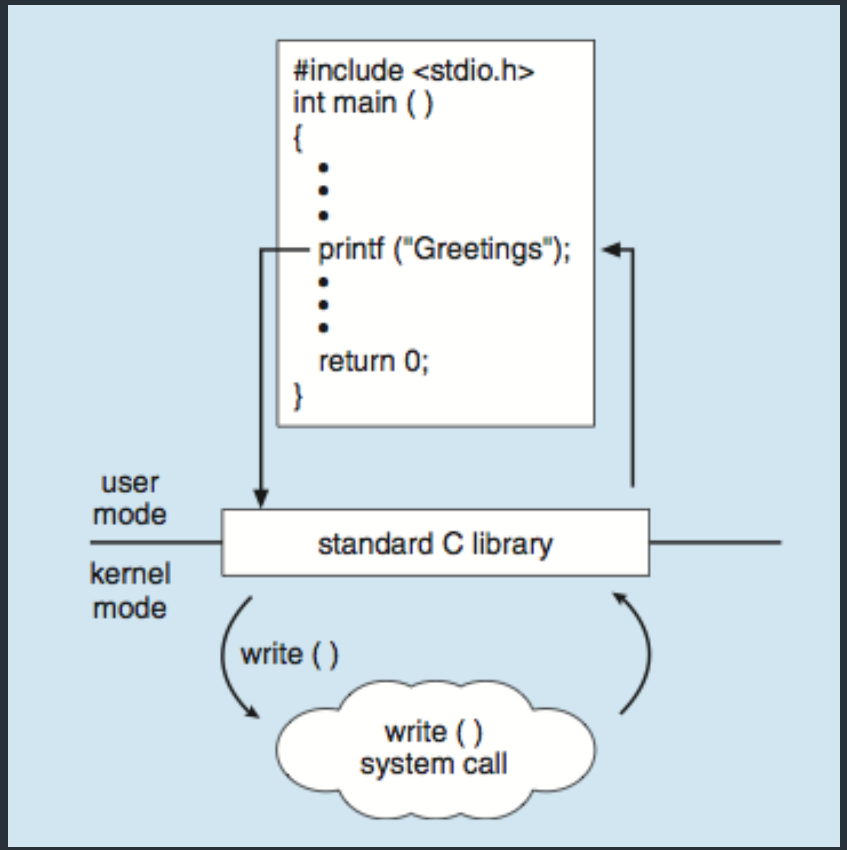


Examples of Windows and Unix System Calls

	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()

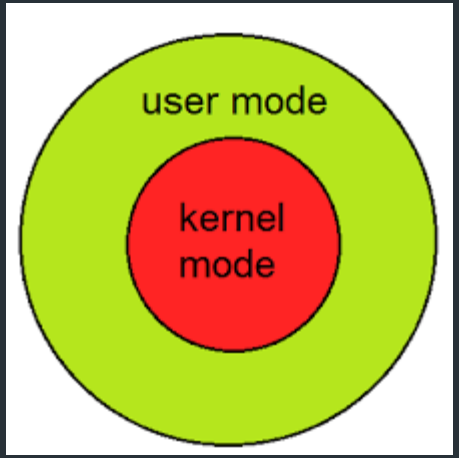
Standard C Library Example

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



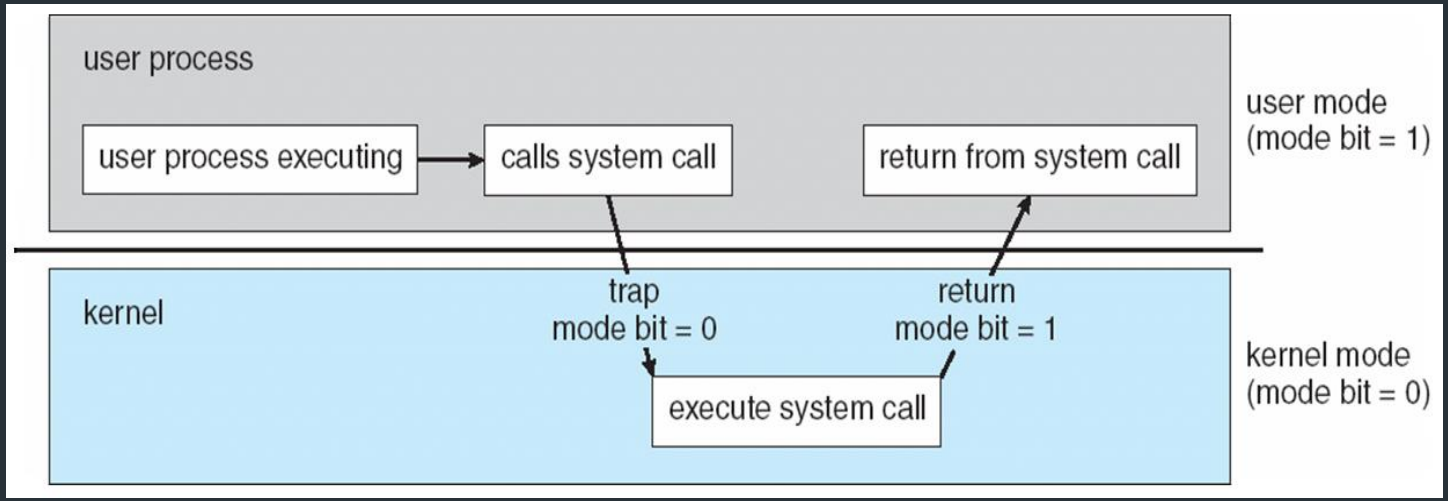
Protection - Modes

- **Dual-mode** operation allows OS to protect itself and other system components
  - **User mode** and **kernel mode**
  - **Mode bit** provided by hardware
    - Provides ability to distinguish when system is running user code or kernel code
    - Some instructions designated as **privileged**, only executable in kernel mode
    - System call changes mode to kernel, return from call resets it to user
- Increasingly CPUs support multi-mode operations
  - i.e. **virtual machine manager** (**VMM**) mode for guest **VMs**



Protection – Modes (Cont.)

- Timer to prevent infinite loop / process hogging resources
  - Timer is set to interrupt the computer after some time period
  - Keep a counter that is decremented by the physical clock.
  - Operating system set the counter (privileged instruction)
  - When counter zero generate an interrupt
  - Set up before scheduling process to regain control or terminate program that exceeds allotted time

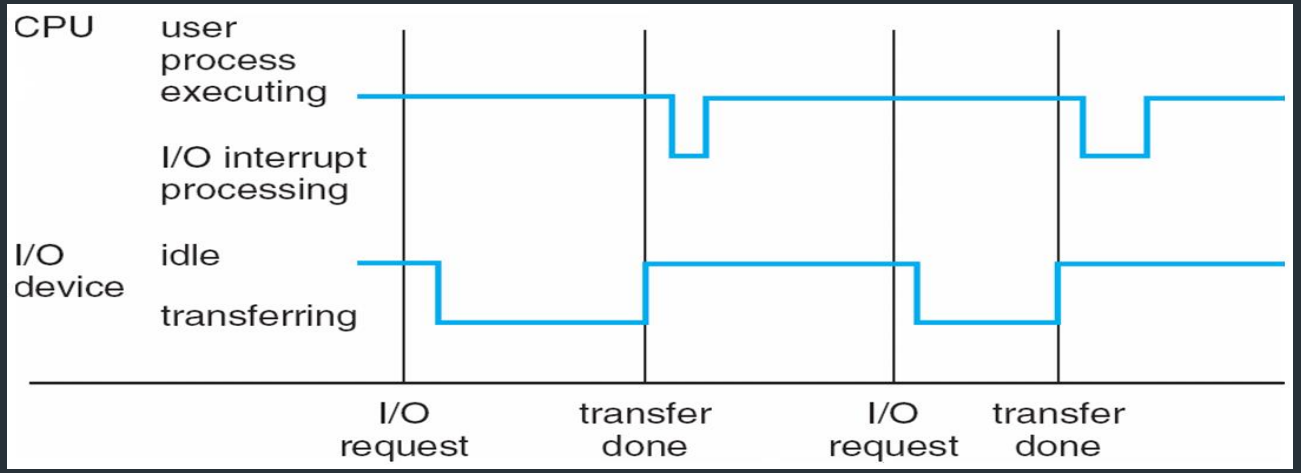


Interrupt

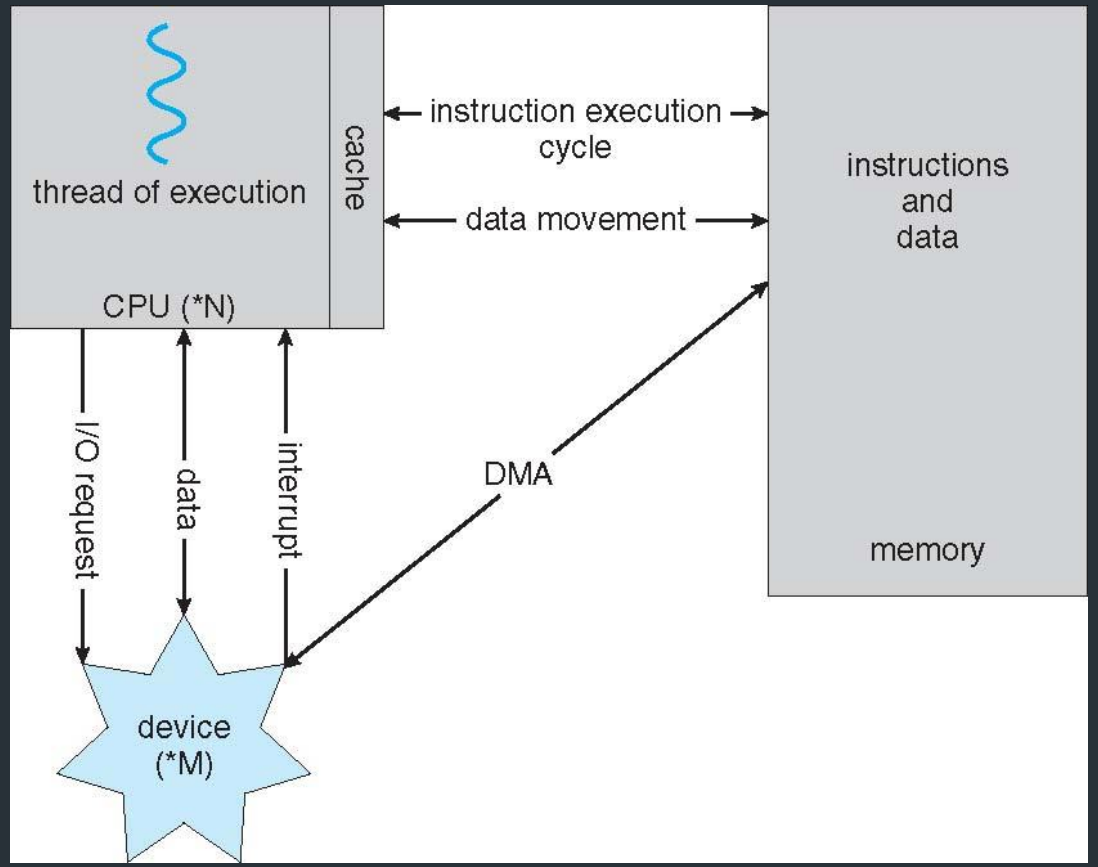
- Interrupt transfers control to the interrupt service routine generally, through the **interrupt vector**, which contains the addresses of all the service routines
- Interrupt architecture must save the address of the interrupted instruction
- A **trap** or **exception** is a software-generated interrupt caused either by an error or a user request
- An operating system is **interrupt driven**
- **Interrupt driven** (hardware and software)
  - Hardware interrupt by one of the devices
  - Software interrupt (**exception** or **trap):**
    - Software error (e.g., division by zero)
    - Request for operating system service
    - Other process problems include infinite loop, processes modifying each other or the operating system

Interrupt Handling

- The operating system preserves the state of the CPU by storing registers and the program counter
- Determines which type of interrupt has occurred:
  - **polling**
  - **vectored** interrupt system
- Separate segments of code determine what action should be taken for each type of interrupt



A von Neumann Architecture



References

- Silberschatz, Gagne, Galvin: Operating System Concepts, 6th Edition
- Remzi H. Arpaci-Dusseau, Andrea C. Arpaci-Dusseau - Operating Systems_ Three Easy Pieces
- Ramez Elmasri, A Carrick, David Levine - Operating Systems_ A Spiral Approach (2009, McGraw-Hill Science_Engineering_Math)
- https://www2.eecs.berkeley.edu/Courses/CS162/


THANK
YOU