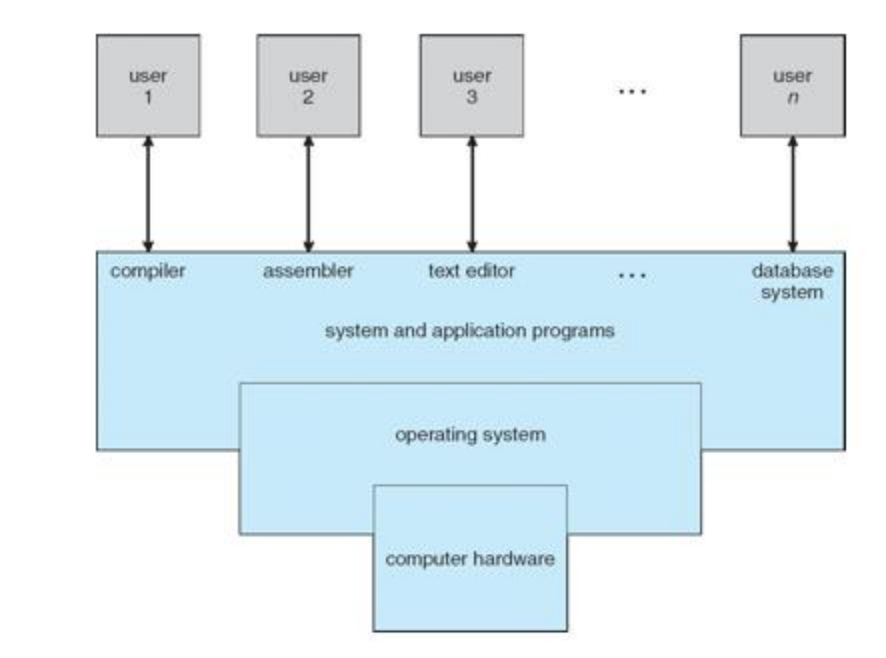
Module 1

Introduction to OS

Dr. Rishikeshan C A
SCOPE, VIT Chennai

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

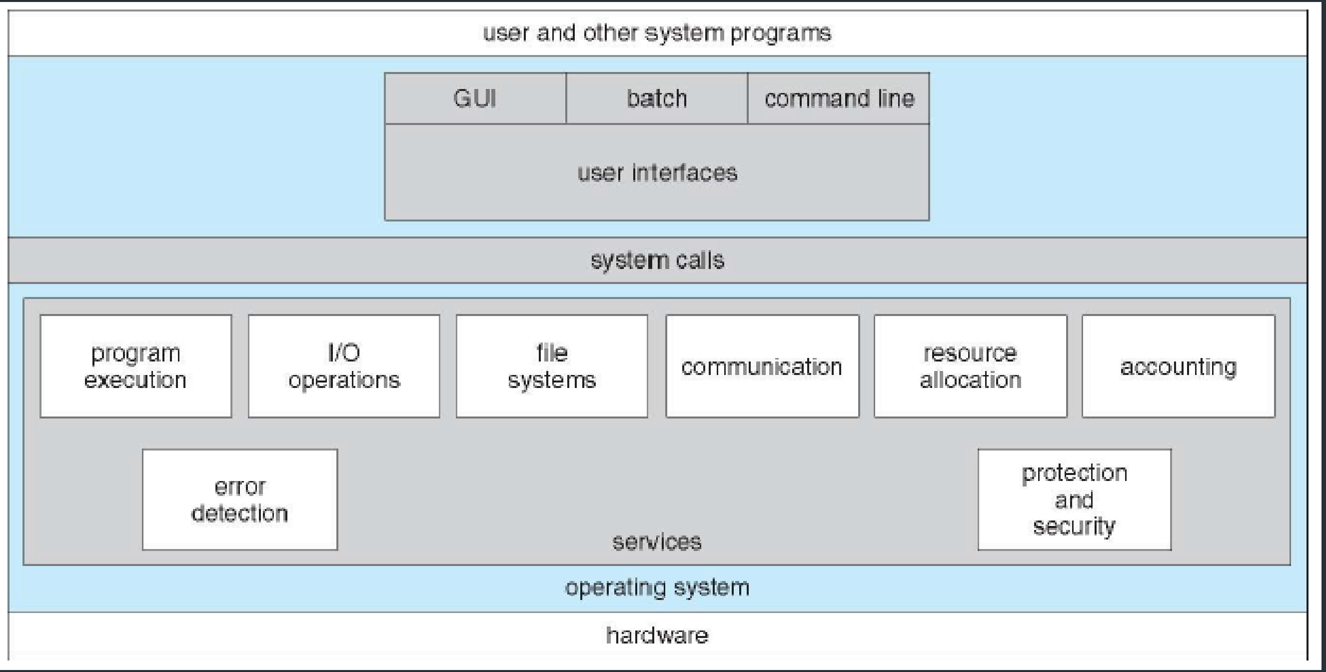
Decides between conflicting requests for efficient and fair resource use

OS is a control program

Controls execution of programs to prevent errors and improper use of the computer

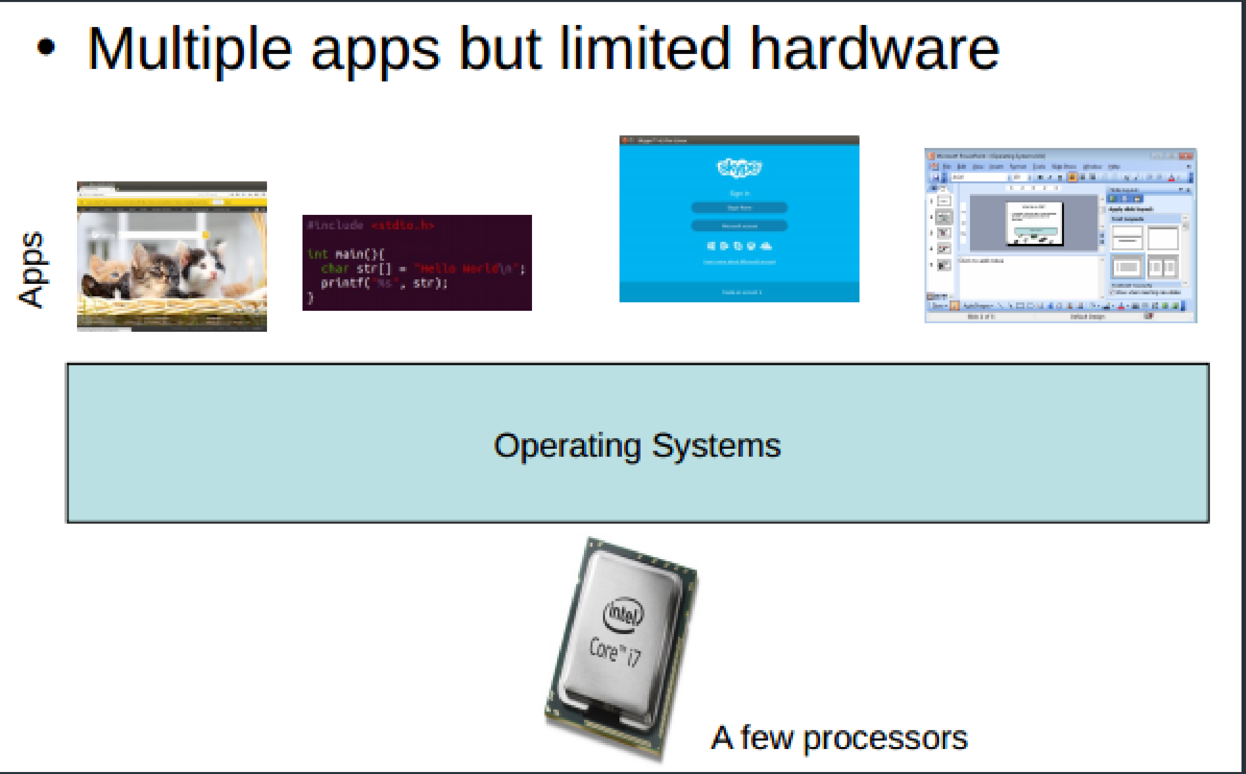
- Application program runs on a platform and that platform is an Operating systems.
- OS plays an important role to determine which application you need, because some applications may exists only in some OS.
- Example:
  - Words in windows
  - Libre office in linux

Schematic of Operating System Services



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

Schematic of Operating System functionality



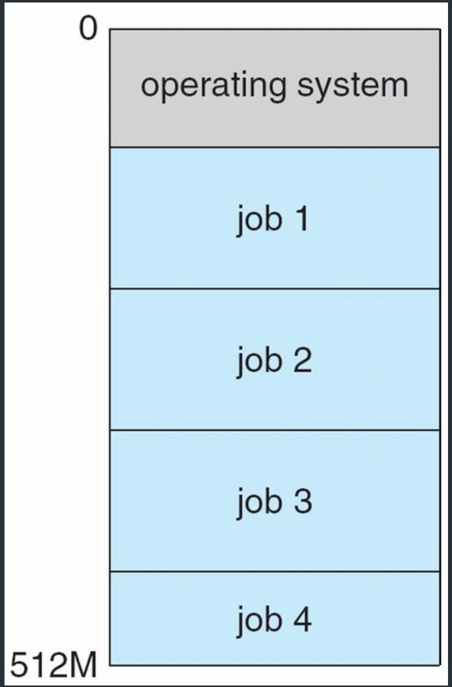
Different types of OS

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

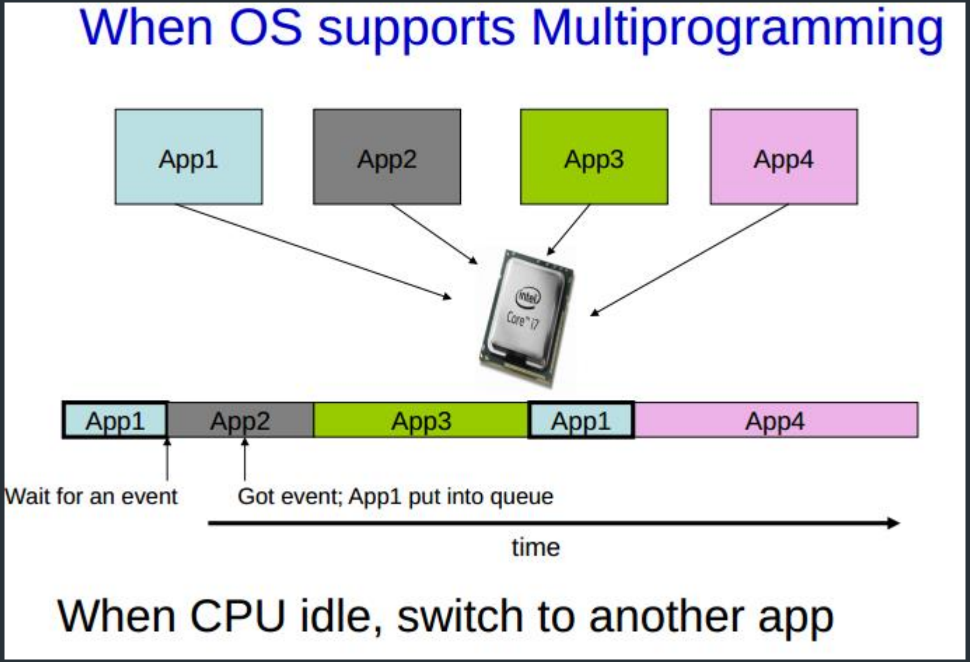
Types of Operating System

- **Multiprogramming (Batch system)** needed for efficiency
  - Single user cannot keep CPU and I/O devices busy at all times
  - Multiprogramming organizes jobs (code and data) so CPU always has one to execute
  - A subset of total jobs in system is kept in memory
  - One job selected and run via **job scheduling**
  - When it has to wait (for I/O for example), OS switches to another job
- **Timesharing (multitasking)** is logical extension in which CPU switches jobs so frequently that users can interact with each job while it is running, creating **interactive** computing
  - **Response time** should be < 1 second
  - Each user has at least one program executing in memory ⇨ **process**
  - If several jobs ready to run at the same time ⇨ **CPU scheduling**
  - If processes don't fit in memory, **swapping** moves them in and out to run
  - **Virtual memory** allows execution of processes not completely in memory

Memory Layout for Multiprogrammed System

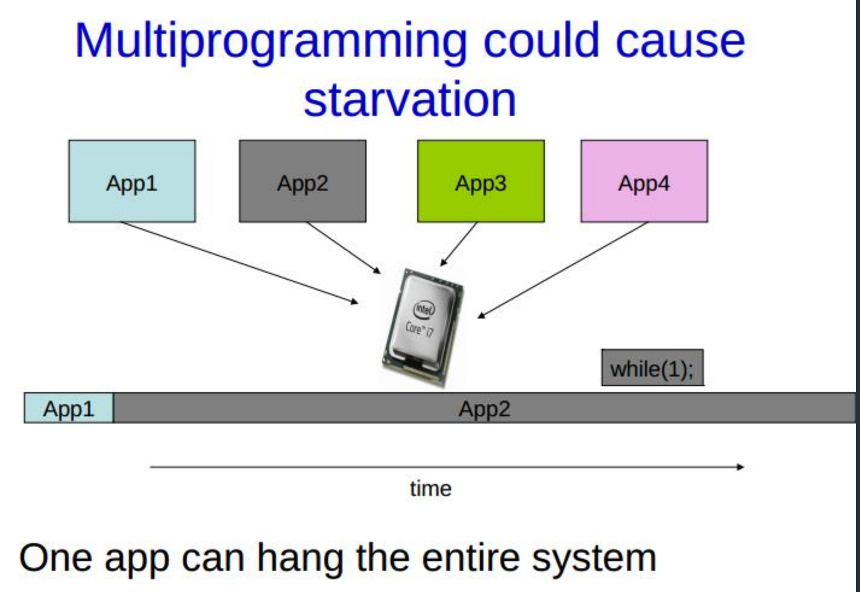


Several jobs are kept in main memory at the same time, and the CPU is multiplexed among them.



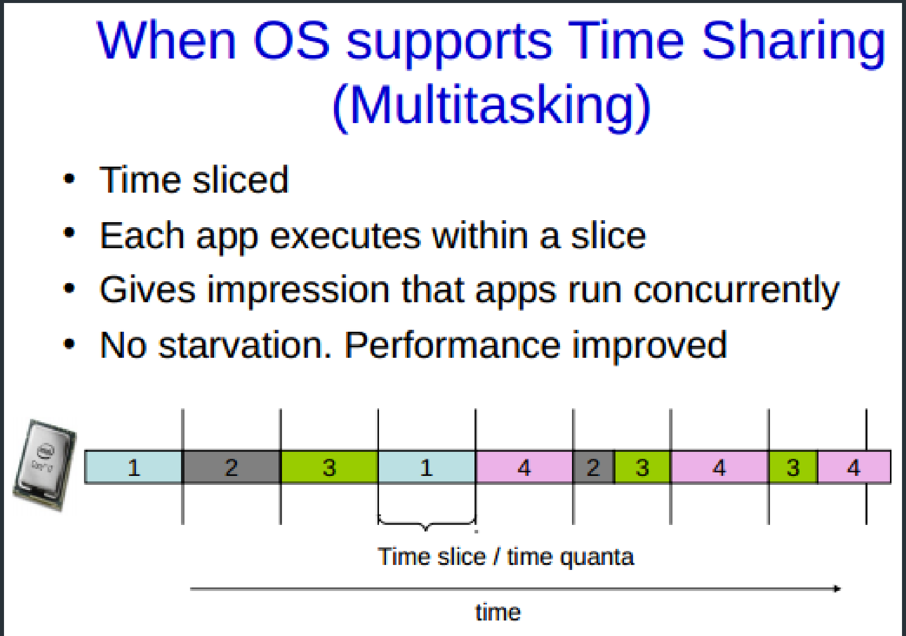
OS Features Needed for Multiprogramming

- I/O routine supplied by the system.
- Memory management – the system must allocate the memory to several jobs.
- CPU scheduling – the system must choose among several jobs ready to run.
- Allocation of devices.



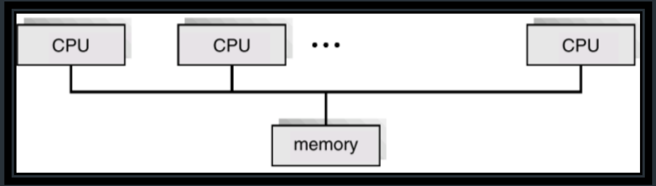
Time-Sharing Systems–Interactive Computing

- The CPU is multiplexed among several jobs that are kept in memory and on disk (the CPU is allocated to a job only if the job is in memory).
- A job swapped in and out of memory to the disk.
- On-line communication between the user and the system is provided; when the operating system finishes the execution of one command, it seeks the next “control statement” from the user’s keyboard.
- On-line system must be available for users to access data and code.



Parallel Systems

- Multiprocessor systems with more than on CPU in close communication.
- *Tightly coupled system* – processors share memory and a clock; communication usually takes place through the shared memory.
- Advantages of parallel system:
  - Increased *throughput*
  - Economical
  - Increased reliability



- *Symmetric multiprocessing (SMP)*
  - Each processor runs and identical copy of the operating system.
  - Many processes can run at once without performance deterioration.
  - Most modern operating systems support SMP
- *Asymmetric multiprocessing*
  - Each processor is assigned a specific task; master processor schedules and allocated work to slave processors.
  - More common in extremely large systems

Distributed Systems

- Distribute the computation among several physical processors.
- *Loosely coupled system* – each processor has its own local memory; processors communicate with one another through various communications lines, such as high-speed buses or telephone lines.
- Advantages of distributed systems.
  - Resources Sharing
  - Computation speed up – load sharing
  - Reliability
  - Communications
- Requires networking infrastructure.
- Local area networks (LAN) or Wide area networks (WAN)
- May be either client-server or peer-to-peer systems.

Real-Time Systems

- Often used as a control device in a dedicated application such as controlling scientific experiments, medical imaging systems, industrial control systems, and some display systems.
- Well-defined fixed-time constraints.
- *Hard real-time system*.
  - Secondary storage limited or absent, data stored in short-term memory, or read-only memory (ROM)
  - Conflicts with time-sharing systems, not supported by general-purpose operating systems.
- *Soft real-time system*
  - Limited utility in industrial control or robotics
  - Useful in applications (multimedia, virtual reality) requiring advanced operating-system features.

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**.

Program execution

- OS handles many activities, that are encapsulated as a process.
- Process refer to a full execution that includes:-
  - code to execute,
  - data to manipulate,
  - registers,
  - OS resources in use.
- When Program is executing the OS manages the following:
  - Loads a program into memory.
  - Executes the program.
  - Handles program's execution.
  - Provides a mechanism for process synchronization.
  - Provides a mechanism for process communication.
  - Provides a mechanism for deadlock handling.

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.

Operating System Structure

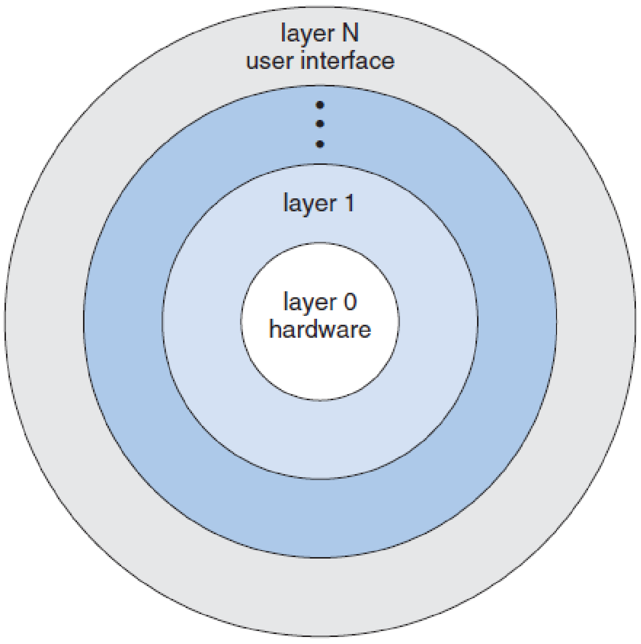
- Monolithic
- Micro-kernel models
- Layered (conceptual)
- Modular

Kernels may be classified mainly in three categories: -

Monolithic Kernel
Micro Kernel
Hybrid Kernel

Layered

- Division into number of layers as shown in figure.
- Innermost layer is hardware
- Outermost layer is interface
- Accordingly as we move from innermost to outermost layer we observe that we are moving from the perspective of hardware to software with each interlinkage layer.
- Simple
- Easy to Debug
- Easy to verify

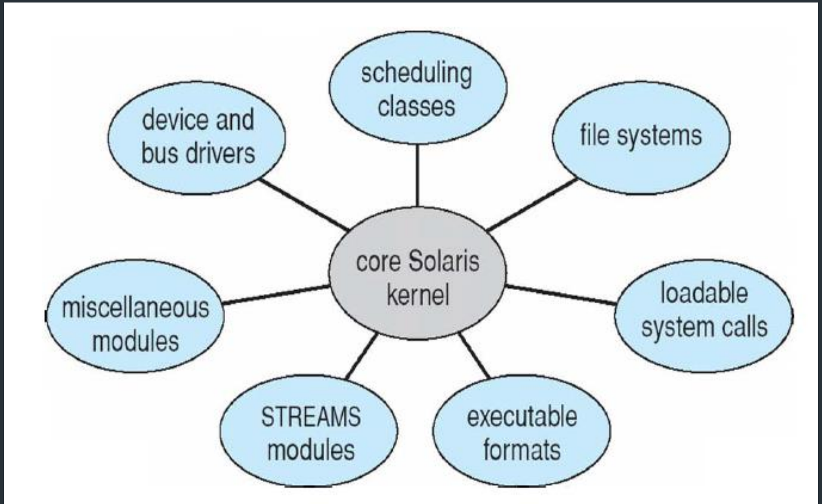


A layered operating system.

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

Modular

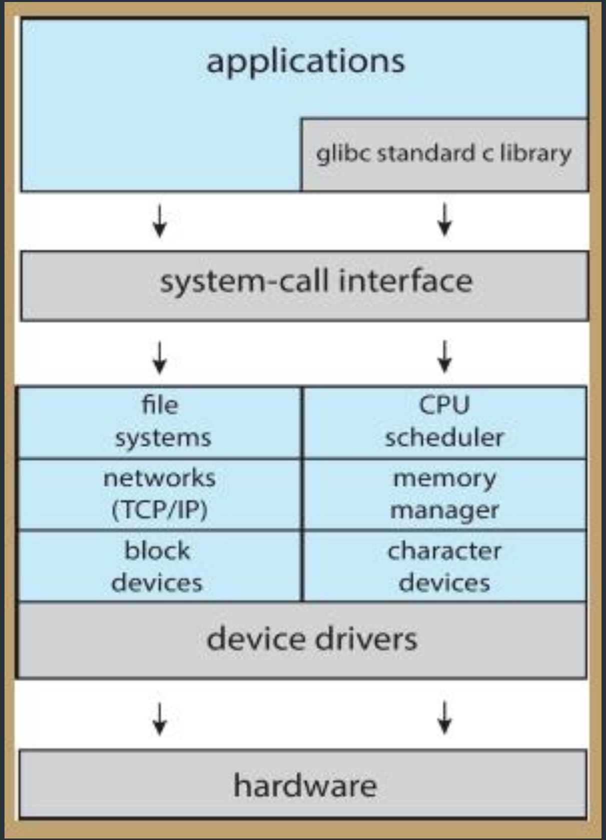
- Divided into different module.
- Typically employs:
  - dynamic loadable kernel module (LKM). i.e. Different modules communicate through kernel (core part).
- LKM may be loaded during boot or when required, and can be deleted also.
- An example would be a device driver support module loaded when a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



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

Monolithic

- Monolithic Kernel
  - value on speed and efficiency.
  - Monolithic is a single static binary file.
  - It executes in a single address space.

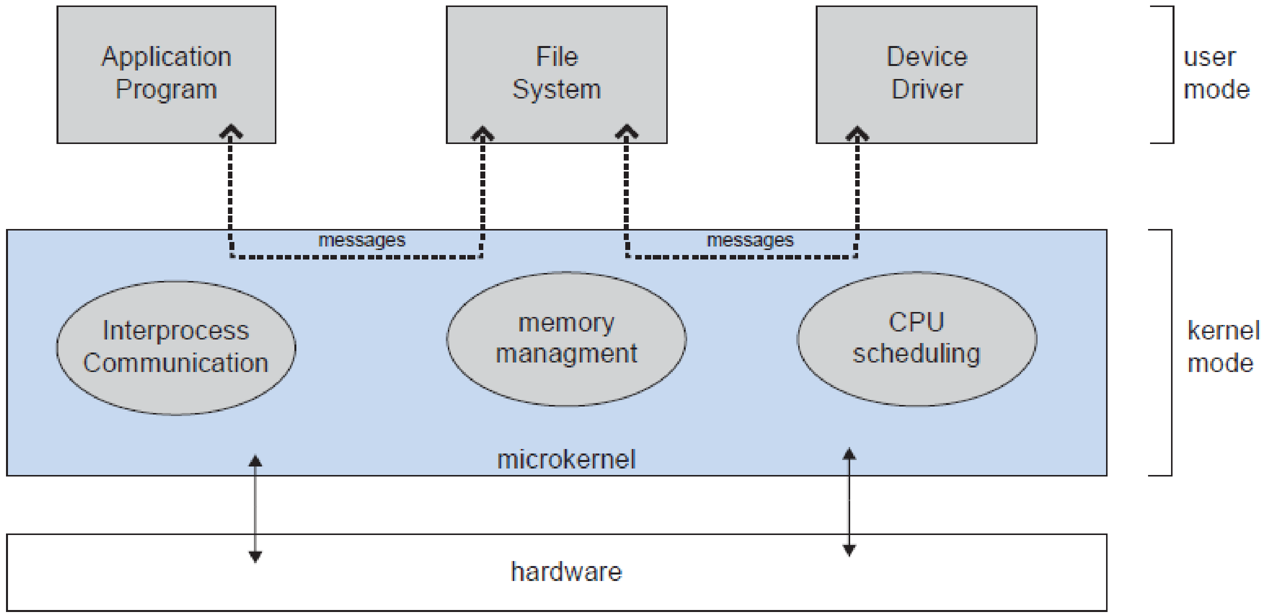


[Silberschatz, Gagne, Galvin: Operating System Concepts, 6th Edition]

Microkernels:

- 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.

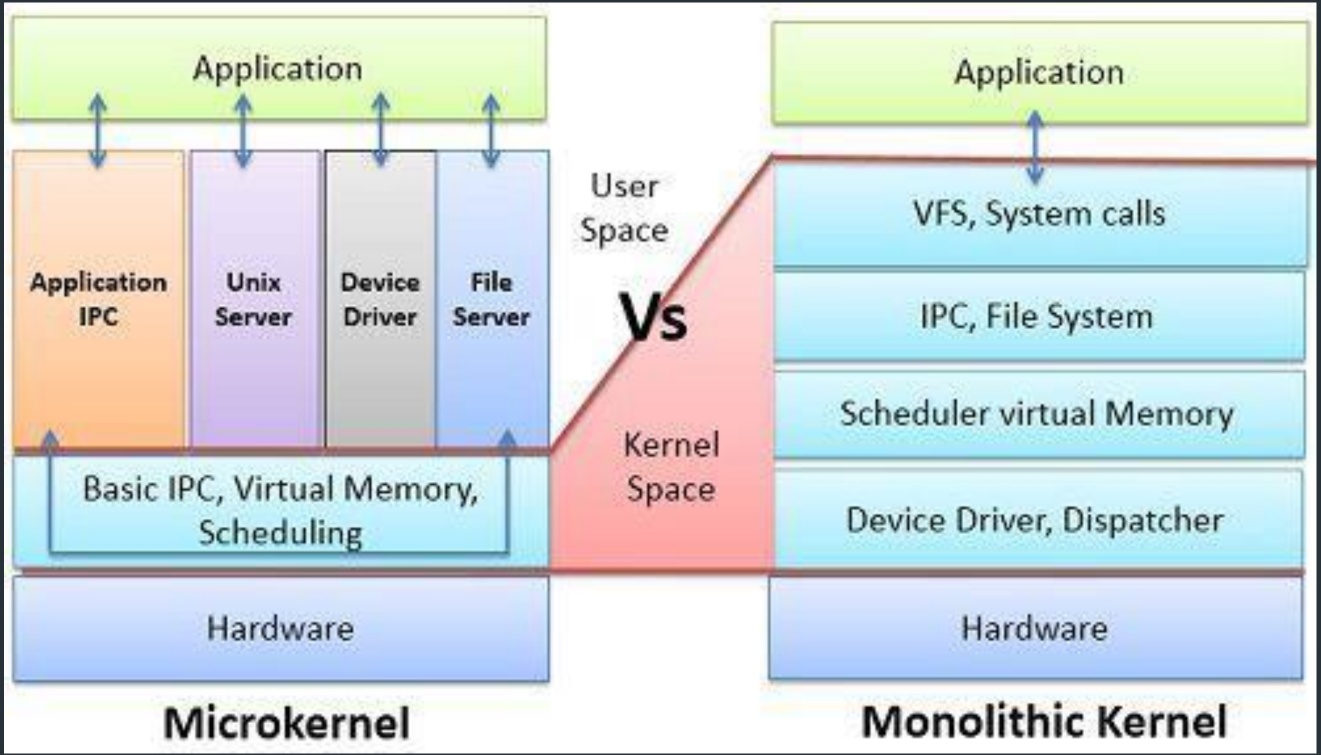
Microkernel



Architecture of a typical microkernel.

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

Monolithic vs Microkernel



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.

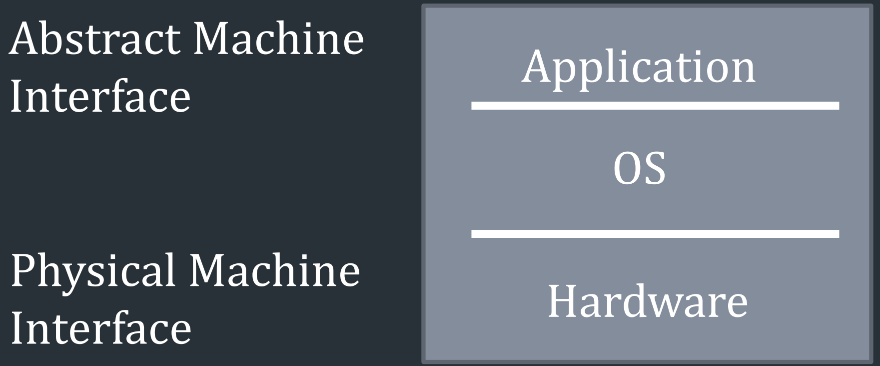
Abstraction

- OS acts as an intermediary between a user and the hardware
- Interface for the user is provided by the OS. This interface is how a user use the service.
- Creates an environment for the user

- **Abstract Machine**
  - Complex details of the hardware are hidden
  - APIs
  - Application development becomes simple
- Command Interpreter
  - Part of a OS that understands and executes commands that are entered interactively by a human being or from a program
  - Shell

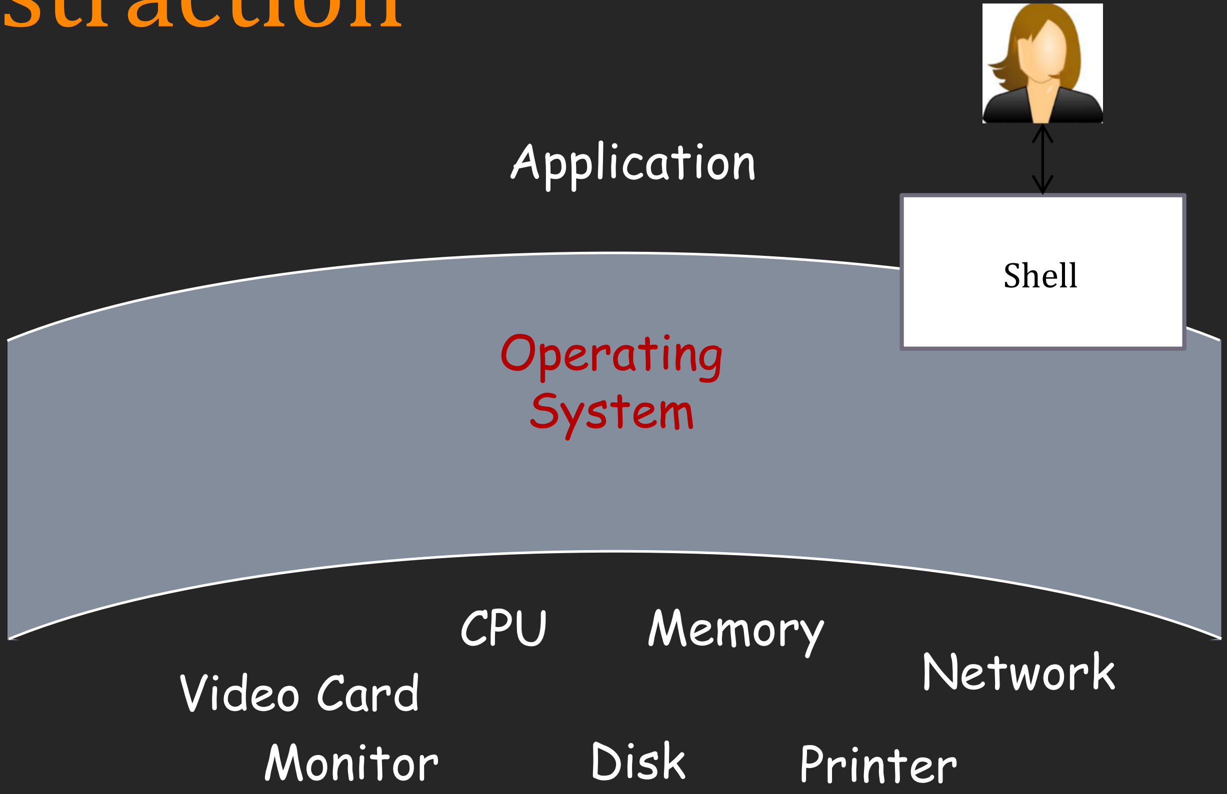
Abstraction

- Processor → Thread
- Memory → Address Space
- Disks, SSDs, … → Files
- Networks → Sockets
- Machines → Processes

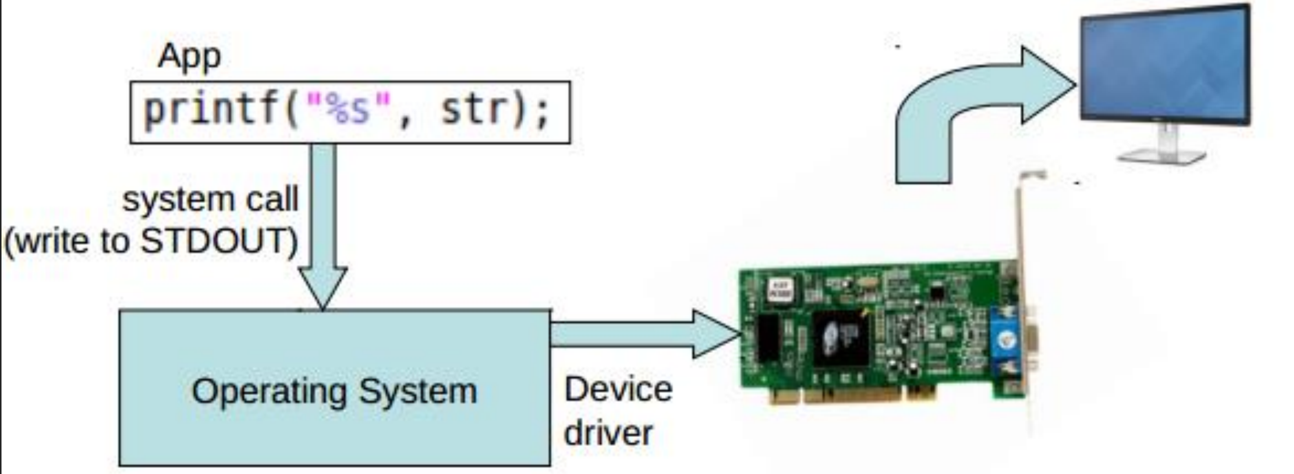


- OS as an Illusionist:
  - Remove software/hardware quirks (fight complexity)
  - Optimize for convenience, utilization, reliability, … (help the programmer)
- For any OS area (e.g. file systems, virtual memory, networking, scheduling):
  - What hardware interface to handle? (physical reality)
  - What's software interface to provide? (nicer abstraction)

Abstraction

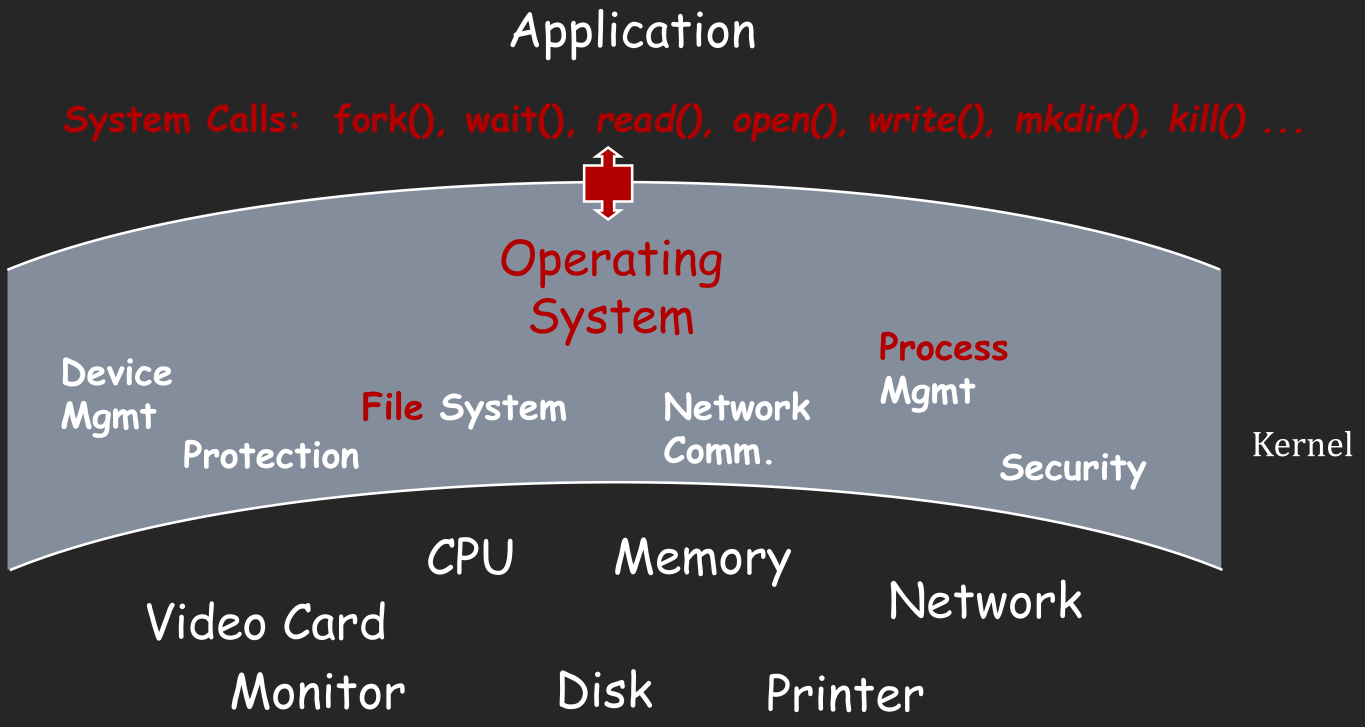


Operating Systems Provide Abstraction



- **Easy to program** apps
  - No more nitty gritty details for programmers
- **Reusable functionality**
  - Apps can reuse the OS functionality
- **Portable**
  - OS interfaces are consistent. The app does not change when hardware changes

Providing abstraction via system calls



Why is abstraction important?

- Without OSs and abstract interfaces, application writers must program all device access directly
  - load device command codes into device registers
  - understand physical characteristics of the devices
- Applications suffer!
  - Very complicated maintenance and upgrading
  - No portability

Concept of Process

- Process
    - Program loaded in memory and in execution.
- Program is a passive whereas process is an active entity

Process Management Activities

The operating system is responsible for the following activities in connection with process management:

- Initiating and Terminating Processes
- To pause and resume processes
- Process scheduling
- Mechanism for:
  - Process synchronization, communication and deadlock handling.

Resource

- OS acts an interface between hardware and software.
- Resources are objects that can be allocated in a computer . Examples:
  - Processors,
  - Devices: Both input and output devices,
  - Memory,
  - Files

- Thus, we can restate the purpose of the operating system in terms of resources.
  - The operating system manages resources (resource allocation) and
  - To provides an interface to resources for application programs (resource abstraction).

Influence of Security

- Security must consider external environment of the system, and protect it from:
- unauthorized access.
- malicious modification or destruction
- accidental introduction of inconsistency.
- Easier to protect against accidental than malicious misuse.

Other Security Issues

Program Threats

Trojans

Trap Door

System Threats

Worms

Viruses

Denial of Services

Networking and OS

- A modern OS contains built-in software designed to simplify networking.
- Typical OS software includes an implementation of TCP/IP and related utility programs such as ping and traceroute, along with device drivers and other software to automatically enable the Ethernet or wireless interface for a device.
- The operating systems of mobile devices normally provide programs to enable Wi-Fi, Bluetooth, and other wireless connectivity.

Multimedia OS

- The operating system provides a comfortable environment for the execution of programs, and it ensures effective utilization of the computer hardware.
- The OS offers various services related to the essential resources of a computer: CPU, main memory, storage and all input and output devices.
- In multimedia applications, a lot of data manipulation (e.g. A/D, D/A and format conversion) is required and this involves a lot of data transfer, which consumes many resources.
- The integration of discrete and continuous multimedia data demands additional services from many operating system components.
- The major aspect in this context is real-time processing of continuous media data and synchronization

Demand on the applications

- Soft real-time applications: statistical guarantees
  - Examples: Streaming media, virtual games
- Interactive applications: no absolute performance guarantees, but low average response times
  - Examples: Editors, compilers
- Throughput-intensive Applications: no performance guarantees, but high throughput
  - Examples: http, ftp servers

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!