Operating System

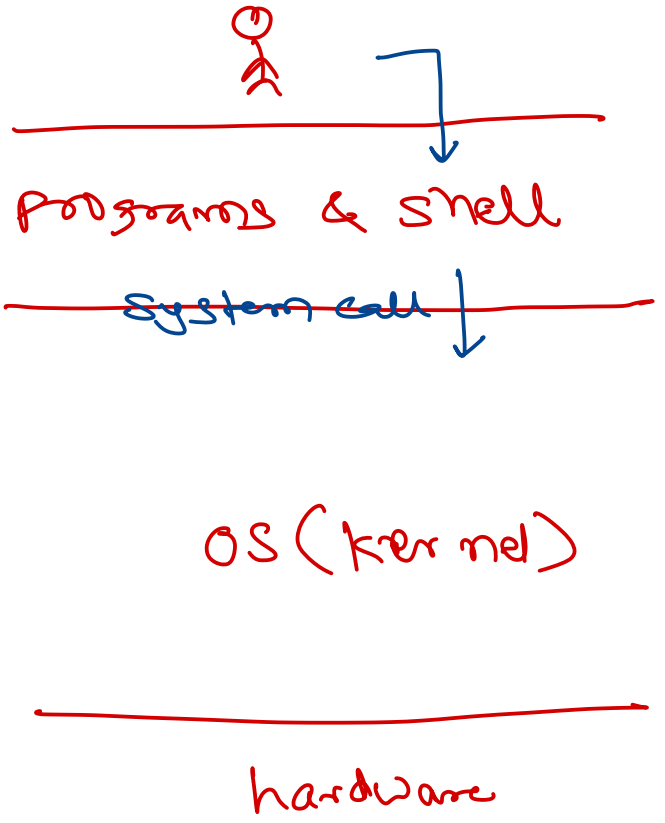
Sunbeam Infotech

What Operating System does?

- There are different types of operating systems, however all these systems have some core services
  - Process Management & cpu scheduling
  - Memory Management
  - Storage Management
  - File System Management
  - I/O System Management & hardware abstraction
  - Protection and Security → anti-virus
  - User interfacing → shell
  - Networking

kernel

extra



shell → Cmnd interpreter.

↳ GUI & CLI

GUI → Windows: explorer.exe

Linux: GNOME or KDE

CLI → windows: cmd.exe

DOS: command.com

Linux: bash, csh, ksh, ...

terminal> echo $SHELL

Process Management

- A process is a program in execution. It is a unit of work within the system. Program is passive, process is active.
- Process needs resources to accomplish its task
  - CPU, memory, I/O, files
  - Initialization data
- Reusable resources are released as process terminates
- Process management activities done by OS:
  - Creating and deleting both user and system processes
  - Suspending and resuming processes
  - Providing mechanisms for process synchronization
  - Providing mechanisms for process communication

Memory Management

- Memory is a large array of words or bytes, each with its own address. It is a quickly accessible by the CPU and I/O devices.
- All data is in memory before and after processing.
- All instructions are in memory for execution.
- Memory management activities
  - Keeping track of which parts of memory are currently being used and by whom
  - Deciding which processes (or parts thereof) and data to move into and out of memory
  - Allocating and deallocating memory space as needed

Storage Management

- Usually disks used to store data that does not fit in main memory or data that must be kept for a “long” period of time
- Most modern computer systems use disks as the principle on-line storage medium, for both programs and data.
- Entire speed of computer operation hinges on disk subsystem and its algorithms
- OS activities
  - Free-space management
  - Storage allocation
  - Disk scheduling
- Some storage need not be fast
  - Tertiary storage includes optical storage, magnetic tape
  - WORM (write-once, read-many-times) and RW (read-write)

File Management

- A file is a collection of related information defined by its creator. Commonly, files represent programs (both source and object forms) and data.
- Files usually organized into directories
- Access control on most systems to determine who can access what
- OS activities include
  - Creating and deleting files and directories
  - Primitives to manipulate files and directories
  - Mapping files onto secondary storage
  - Backup files onto stable (non-volatile) storage media

I/O Management

- One purpose of OS is to hide peculiarities of hardware devices from the user
- I/O subsystem responsible for
  - Memory management of I/O including buffering (storing data temporarily while it is being transferred), caching (storing parts of data in faster storage for performance), spooling (the overlapping of output of one job with input of other jobs)
  - General device-driver interface
  - Drivers for specific hardware devices

Protection and Security

- Protection is mechanism for controlling access of processes or users to resources defined by the OS.
  - Systems generally first distinguish among users, to determine who can do what
    - User identities (user IDs, security IDs) include name and associated number, one per user
    - User ID then associated with all files, processes of that user to determine access control
    - Group identifier (group ID) allows set of users to be defined and controls managed, then also associated with each process, file
    - Privilege escalation allows user to change to effective ID with more rights
- Security is defense of system against internal/external attacks
  - Huge range, including denial-of-service, worms, viruses, identity theft, theft of service

Operating System Interface - CLI

- User communicates with the core OS graphical or command line interface.
- CLI allows direct command entry
- Sometimes implemented in kernel, sometimes by systems program
- Sometimes multiple flavors implemented – shells
  - Primarily fetches a command from user and execute it
    - Sometimes commands built-in, sometimes just names of programs
    - If the latter, adding new features doesn't require shell modification

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
- Many systems now include both CLI and GUI interfaces
  - Microsoft Windows is GUI with CLI “command” shell
  - Apple Mac OS X as “Aqua” GUI interface with UNIX kernel underneath and shells available
  - Solaris is CLI with optional GUI (Java desktop, KDE)

Operating System Design

Sunbeam Infotech

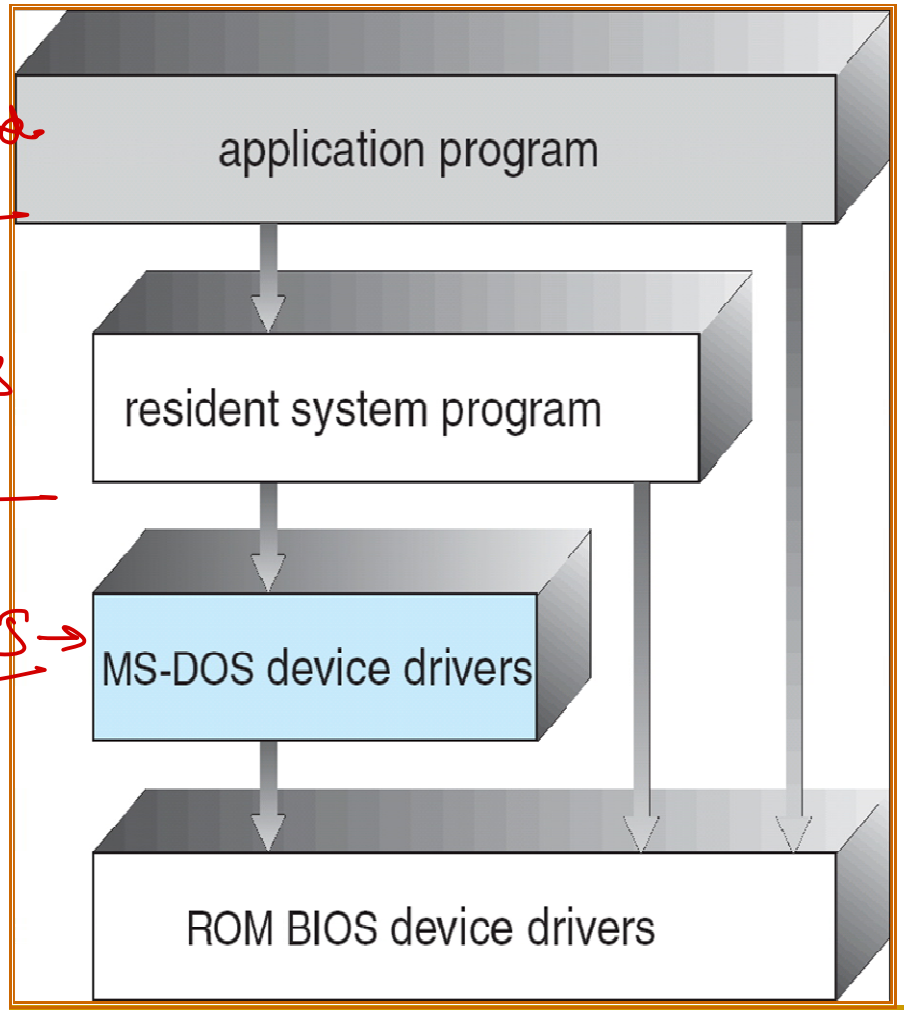
OS Design and Implementation

- Design and Implementation of OS not "solvable", but some approaches have proven successful
- Internals of different Operating Systems can vary widely
- Start by defining goals and specifications
- Affected by choice of hardware, type of system
  - User goals and System goals
  - User goals – operating system should be convenient to use, easy to learn, reliable, safe, and fast
  - System goals – operating system should be easy to design, implement, and maintain, as well as flexible, reliable, error-free, and efficient

OS Design and Implementation

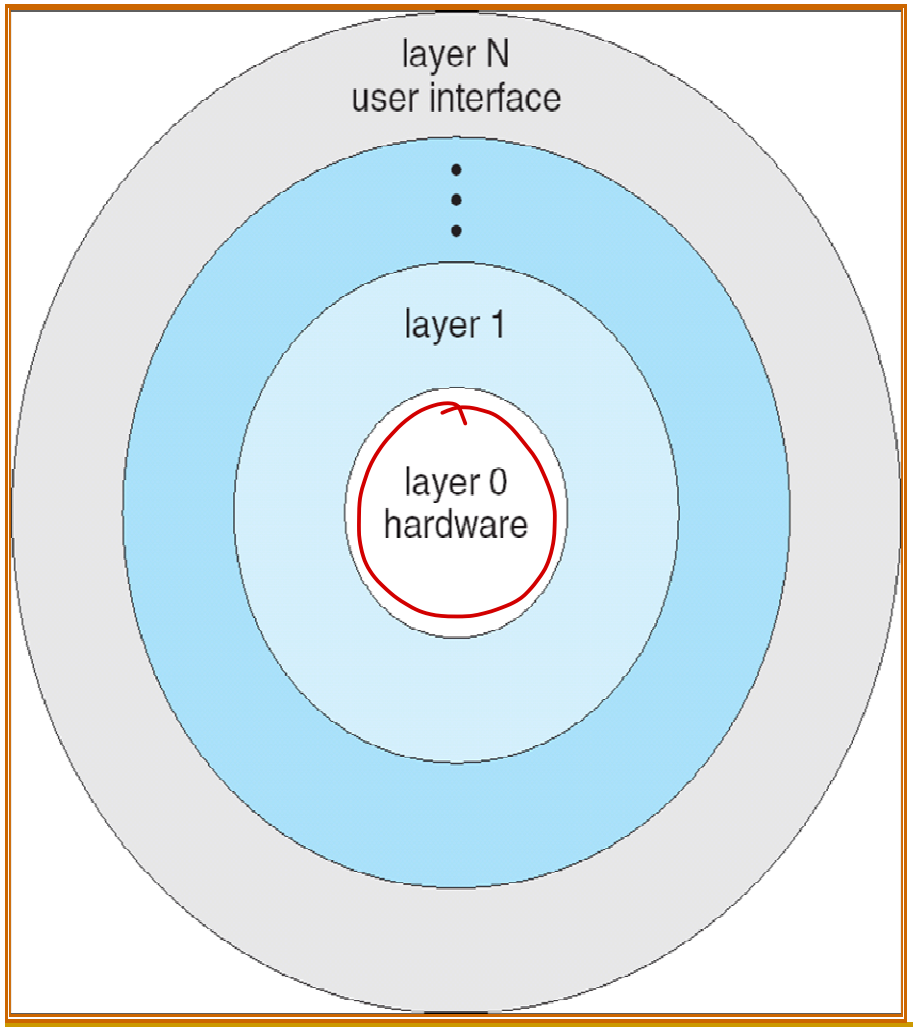
- Important principle to separate
  - Policy: What will be done?
  - Mechanism: How to do it?
- Mechanisms determine how to do something, policies decide what will be done.
- The separation of policy from mechanism is a very important principle, it allows maximum flexibility if policy decisions are to be changed later
- There are different examples of operating systems depending on the goals of designs and the policies and mechanisms.

Simple Structure : MS-DOS



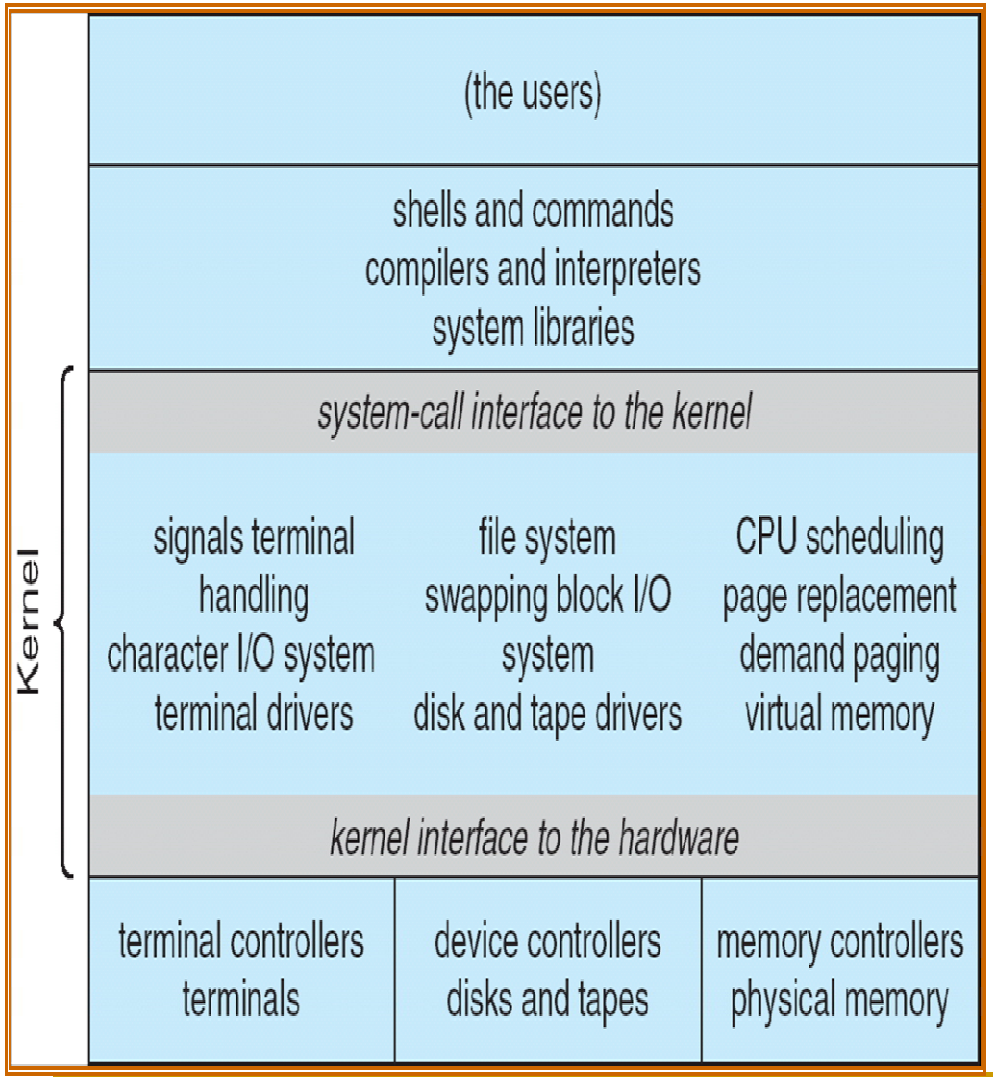
- MS-DOS – written to provide the most functionality in the least space
  - Not divided into modules
  - Although MS-DOS has some structure, its interfaces and levels of functionality are not well separated

Layered Structure



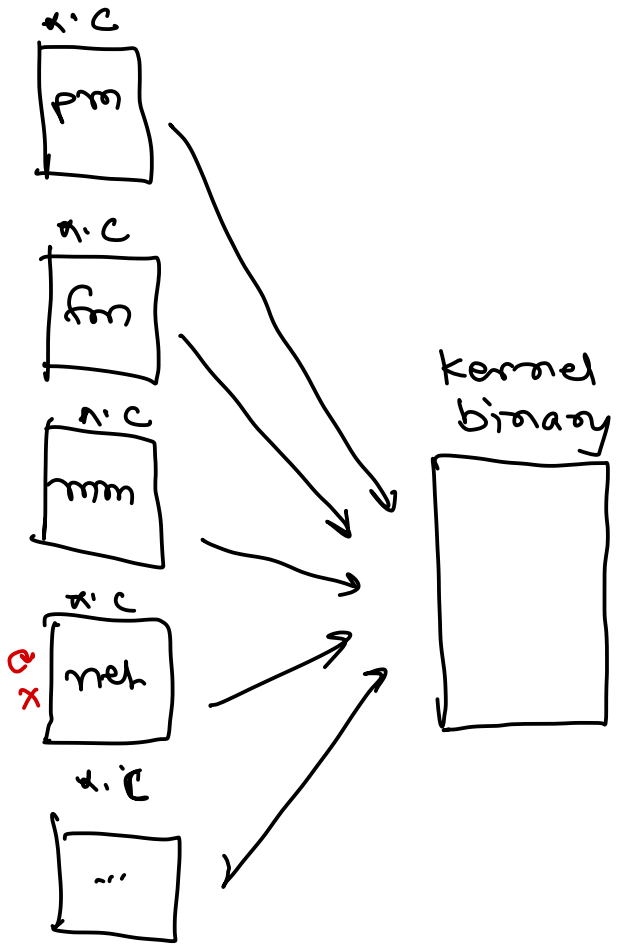
- The operating system is divided into a number of layers (levels), each built on top of lower layers. The bottom layer (layer 0), is the hardware; the highest (layer N) is the user interface.
- With modularity, layers are selected such that each uses functions (operations) and services of only lower-level layers

UNIX System



- UNIX is limited by hardware functionality, the original UNIX had limited structuring.
- UNIX consists two separate parts
  - Systems programs
  - The kernel
    - Consists of everything below the system-call interface and above the physical hardware
    - Provides the file system, CPU scheduling, memory management, and other operating-system functions; a large number of functions for one level

Monolithic



✓ ① fast (all modules are in same binary).

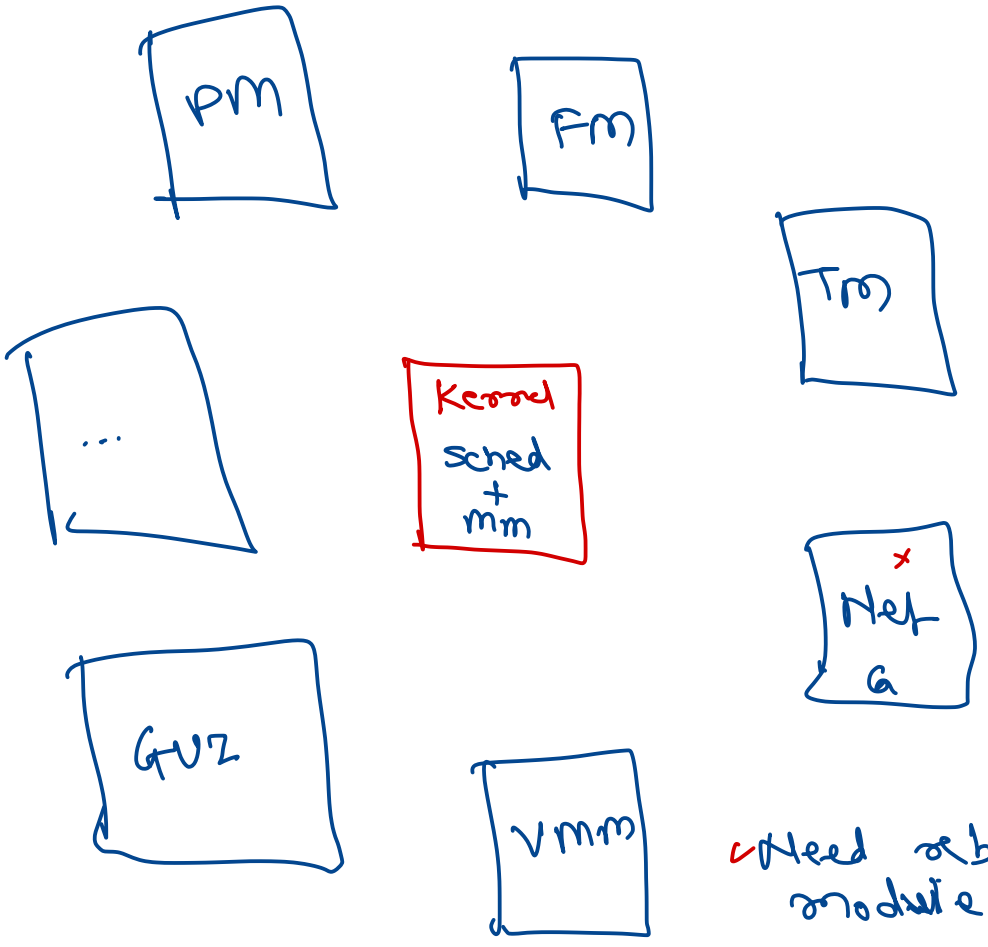
✗ ② if any module fails, kernel crash.

✗ ③ any change in any src code, force to re build & re deploy whole kernel.

DOS, UNIX, Linux (?)

↳ /boot/vmlinuz

Micro-kernel



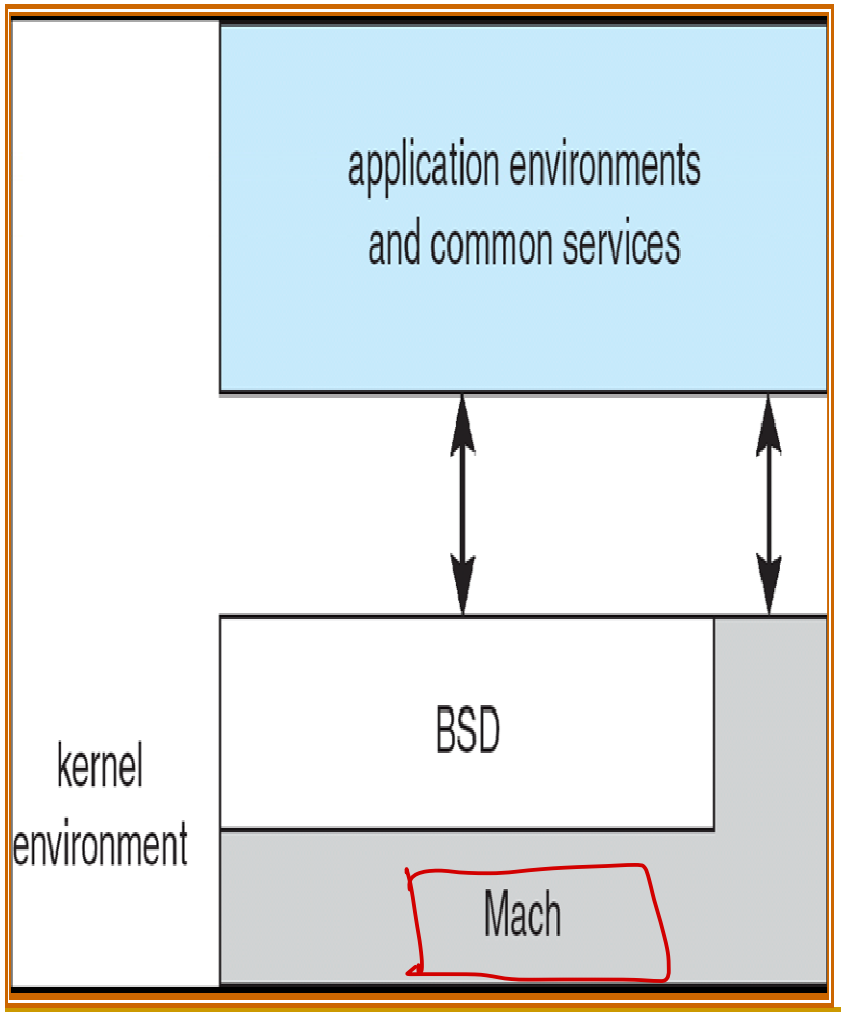
'independent processes provide diff fcns. They are called "servers".

Multiple servers communicate with each other using IPC (msgs are passing).

- if a module fail, only that fcn stop. eg. A sys is waiting.
- Need rebuild only the modified module.

x slower execution (IPC)

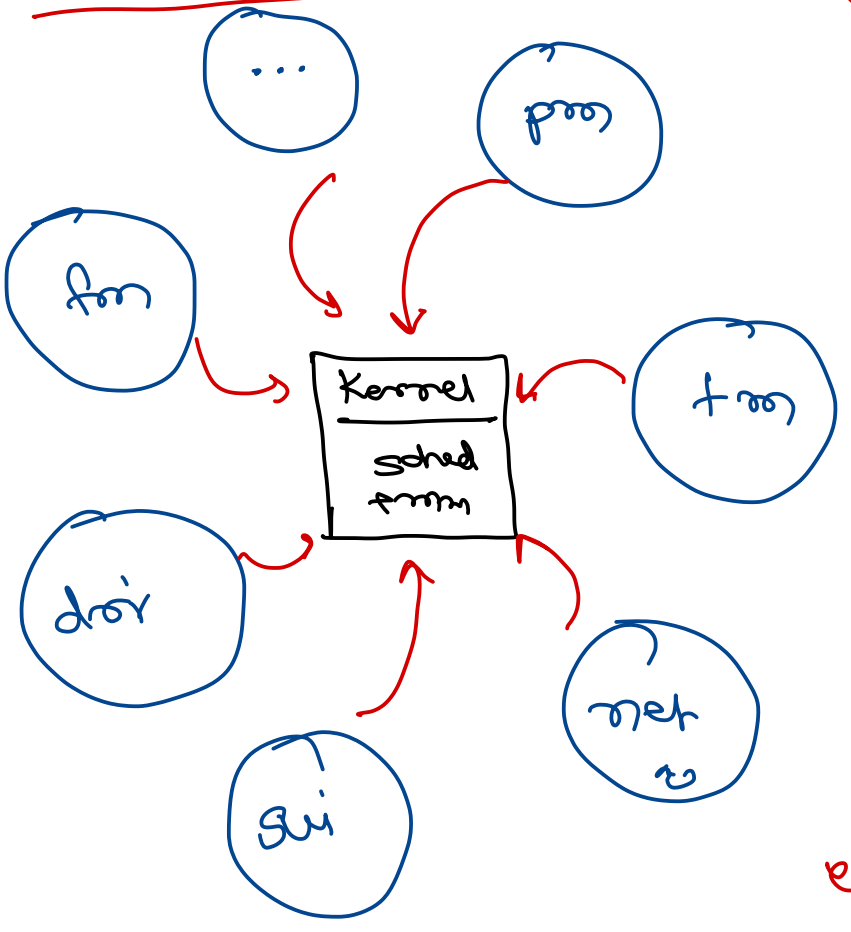
Microkernel Structure: MAC OS X



- Moves as much from the kernel into "user" space
- Communication takes place between user modules using message passing
  - Easier to extend a microkernel
  - Easier to port on new architecture
  - More reliable (less code running in kernel mode)
  - More secure
  - Performance overhead of user space to kernel space comm^{n}

Symbian, QNX,

Modular Kernel



(win) .dll or (linux) .so

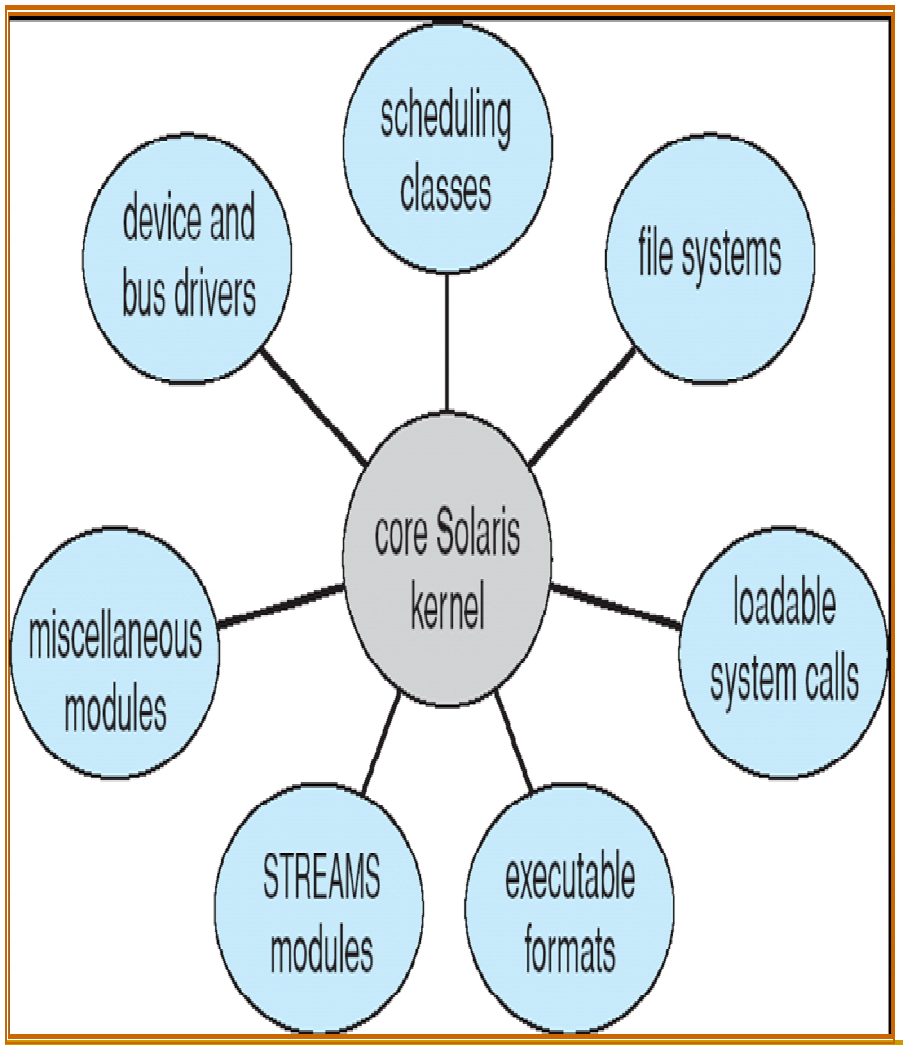
+ fast (all modules are loaded in same process)

+ any change in component need to rebuild that component only.

x if module fails, kernel crash.

e.g. windows: kernel = ntoskrnl.exe

Modular Structure: Solaris



- Most modern operating systems implement 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
- Overall, similar to layers but with more flexible

Hybrid Kernel → made up of multiple kernels.

Mac OS X = BSD UNIX + Mach

BSD UNIX → Monolithic

Mach → Microkernel

1. monolithic
2. microkernel
3. modular
4. hybrid

Linux Kernel

1. **Static Components**
   - Process, Thread, Scheduler
   - Syscalls, Memory, ...

   } vmlinuz ↑ Kernel

2. **Dynamic Components**
   - Device Drivers
   - file systems

   } .ko → modules

UNIX

1965: AT&T, MIT, GE
↓
MULTICS

Dennis Ritchie
+ Ken Thompson → UCB → BSD UNIX

1968 : Project stopped

1969 : AT&T : office editorial system

1970 : UNIX → PDP-7
PL/1 + Assembly

1971: C programming lang.

1972: UNIX – PDP-11
C + assembly

Bill Joy
1. vi editor
2. c shell
3. n-curses lib
4. virtual memory
5. demand paging
6. TCP protocol
7. Socket

UNIX flavours

AT&T → UNIX

USB → BSD UNIX

Sun → Solaris

HP → HP-UX

DG → DG-UX

SCO → SCO UNIX

SGI → IRIX (OpenGL)

MS → Xenix (→ PC)

Apple → Mac OS X

IBM → AIX

Comer → XINU

Tenanbaum → minix

Linus Torvalds → Linux

Richard Stallman → OSF

www.kernel.org

Linux Distributions

www.kernel.org

Redhat → Kernel + GUI + BL + Utils + apps → RHEL, Fedora → CentOS

Novell → SuSE

Debian → Ubuntu → Mint

Slackware

Knoppix

Backtrack

www.distrowatch.com

Thank you!

Source: Galvin OS books/slides

Edited by: Nilesh Ghule