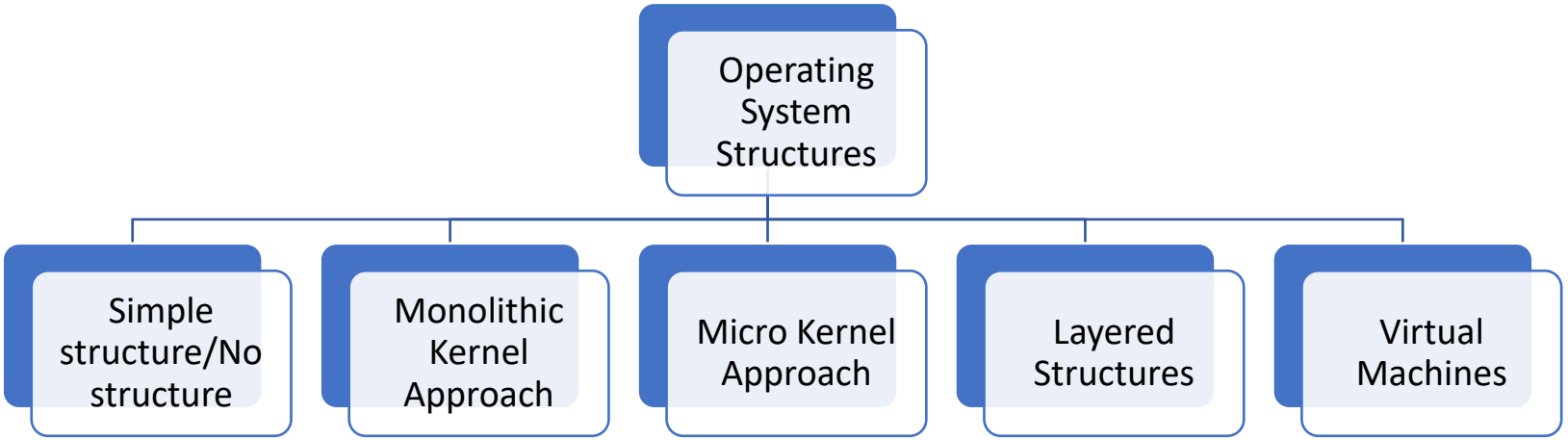
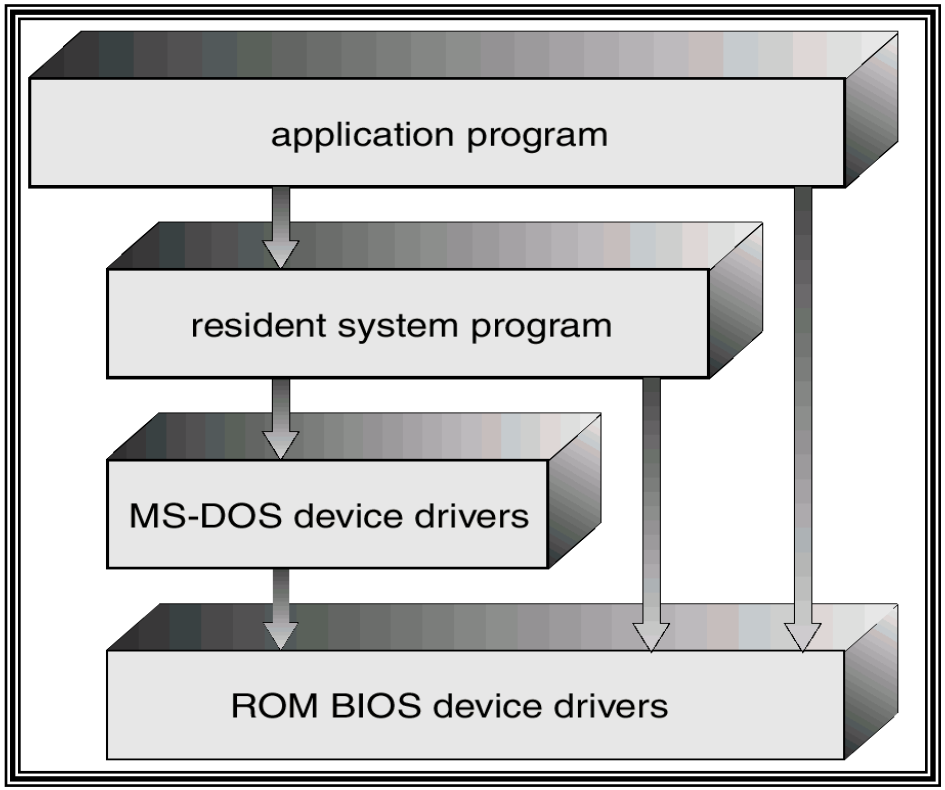
Operating Systems

OS Structures

- **The design process itself requires a highly structured, modularized (even object oriented) approach**
- **Need the ability to allow hundreds of people to work on the system at one time in an orderly way**
- **System must be testable and verifiable**
- **Need the ability to maintain the system:**
  - ✓ **Fix bugs**
  - ✓ **Add new features**
  - ✓ **Tune/customize the system**
- **Need to get the system out on time with "no bugs" … remember Windows 95!**
  - ✓ **System must be bullet-proof**
- **Systematic OS design approach would support the above requirements.**



- 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

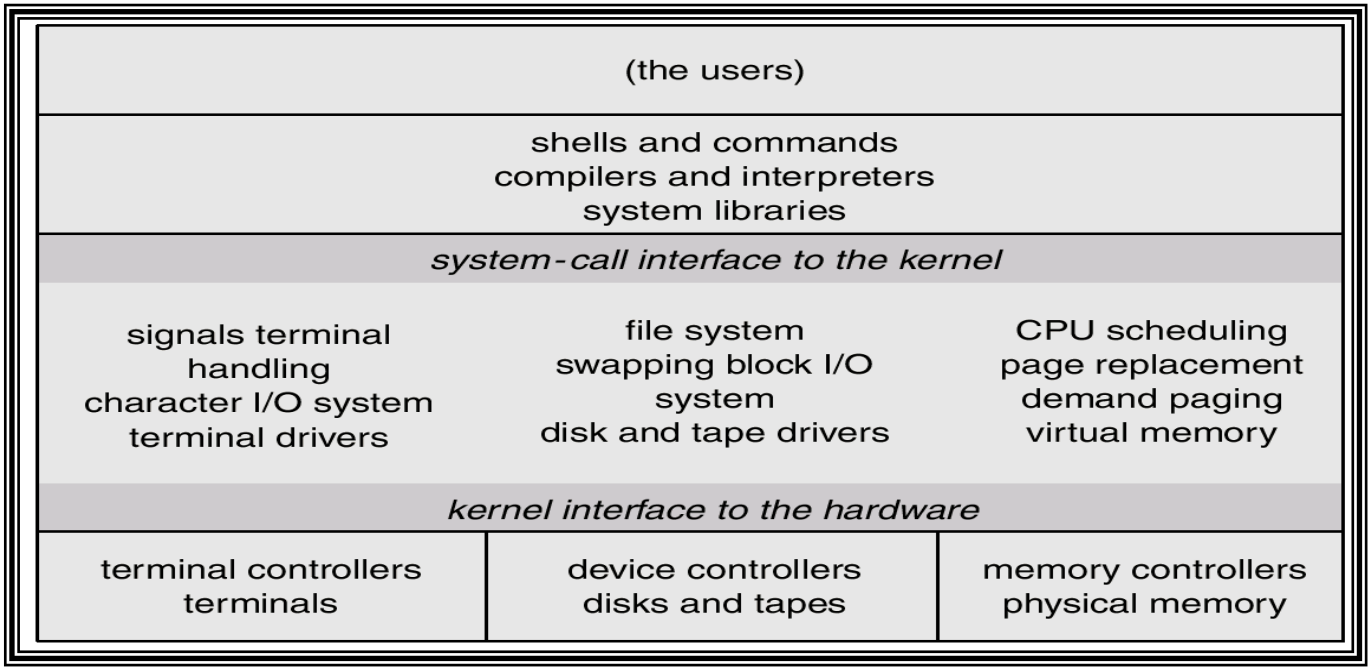


•UNIX – limited by hardware functionality, the original UNIX operating system had limited structuring. The UNIX OS consists of two separable parts.

- ✓ 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.

Layered structure

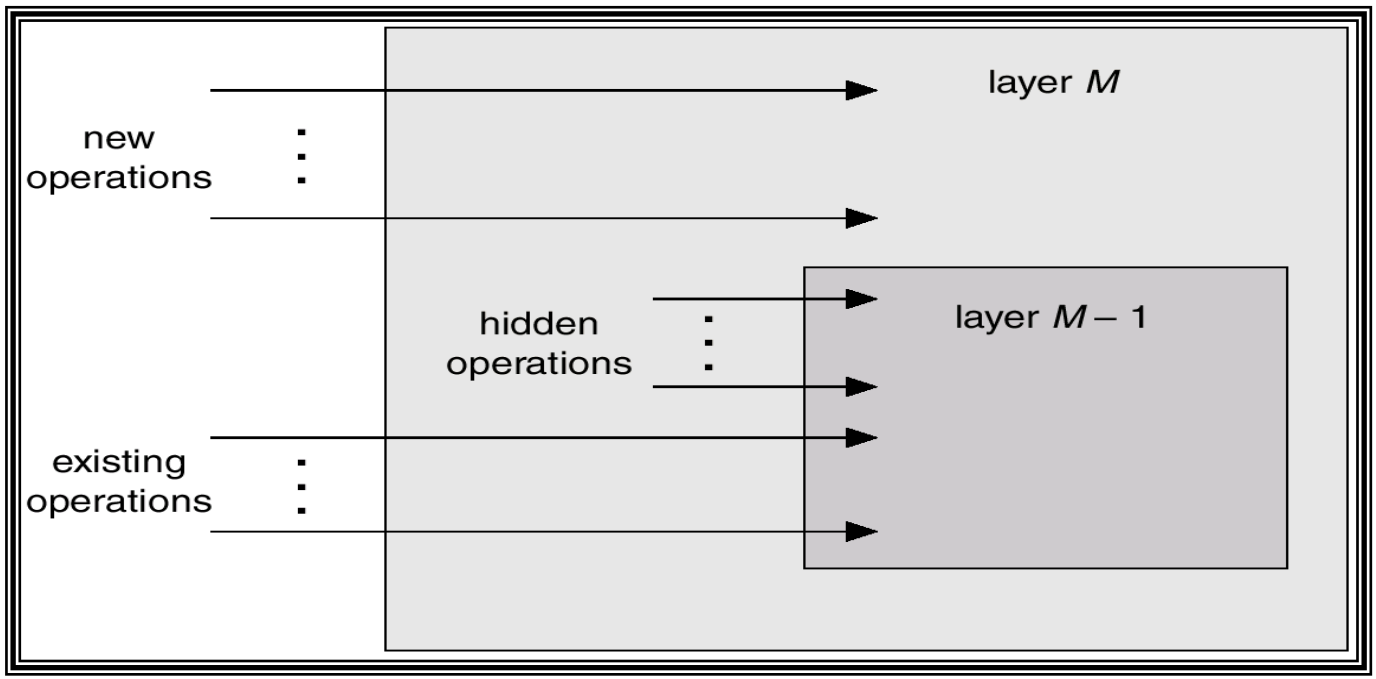
Kernel ➔



apps

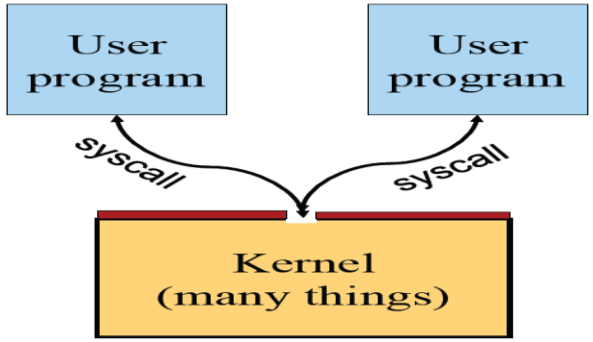
System programs

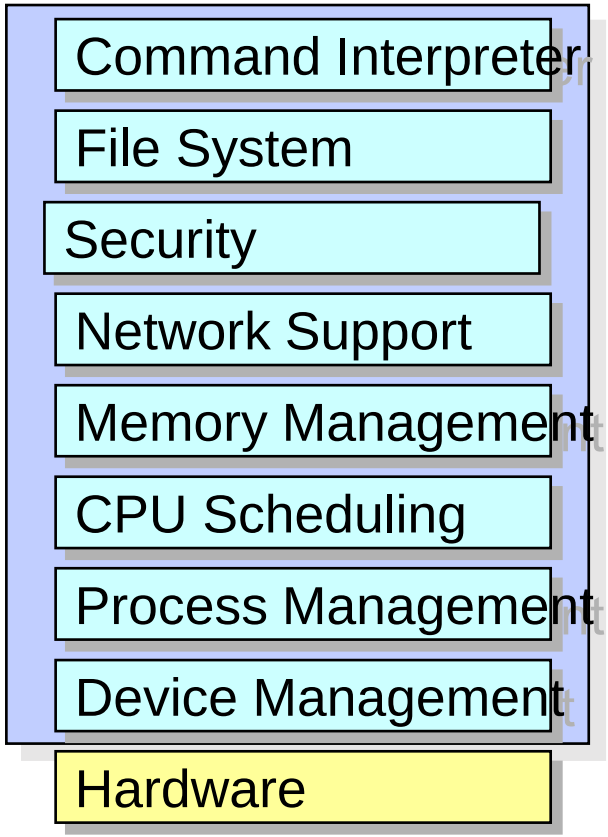
- The operating system is divided into a number of layers (levels), each built on top of lower layers. The bottom layer (layer 0), is the hardware; the highest (layer N) is the user interface.
- With modularity, layers are selected such that each uses functions (operations) and services of only lower-level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



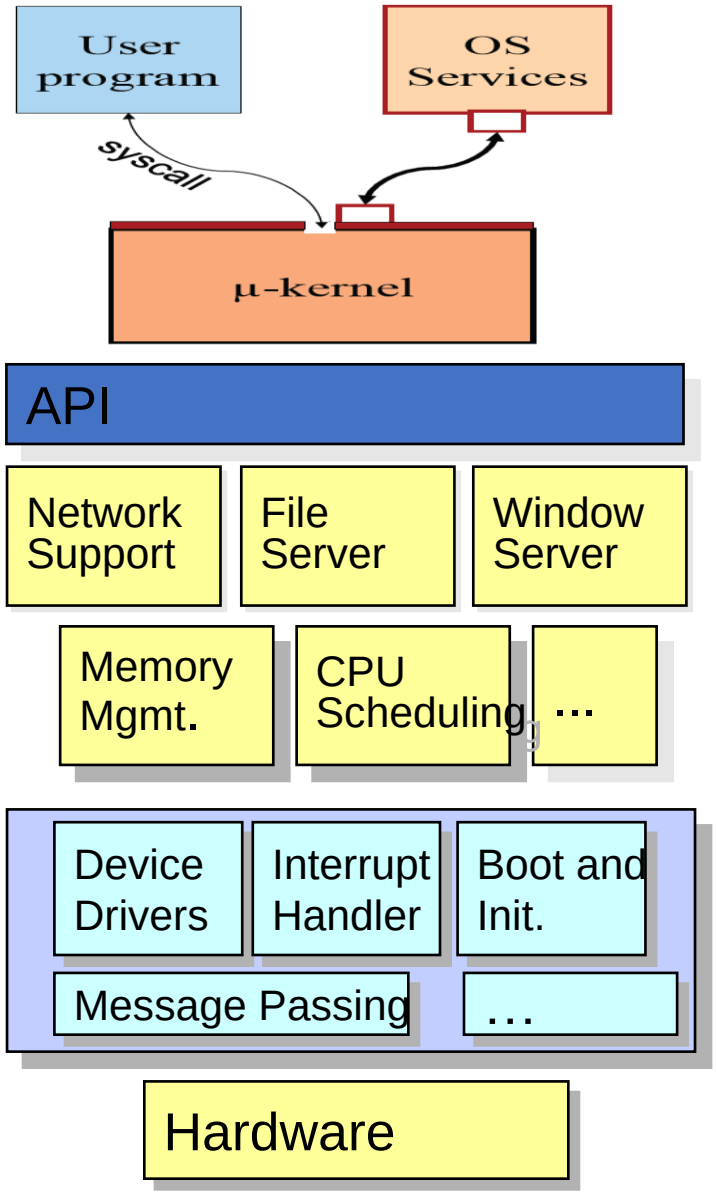
- Examples:
  - ✓ THE (6 layers)
  - ✓ MULTICS (8 rings)
- Pros
  - ✓ Layered abstraction
  - ✓ Separation of concerns
  - ✓ Elegance
  - ✓ Aids in maintainability, and ease and speed of development.
  - ✓ Allows reusable code.
- Cons
  - ✓ Protection boundary crossings
  - ✓ Performance
  - ✓ Inflexible

- All kernel routines are together, any can call any
- A system call interface
- Examples:
  - ✓ Linux, BSD Unix, Windows
- Pros
  - ✓ Shared kernel space
  - ✓ Good performance
- Cons
  - ✓ No information hiding
  - ✓ Chaotic
  - ✓ Hard to understand
  - ✓ How many bugs in 5M lines of code?





- **Strip down the kernel: minimal process, memory, management and communication facilities.**
- Moves as much from the kernel into "*user*" space.
- Communication **between services running in user space** takes place between user modules using message passing. **Messages go thru microkernel.**
- **All new services to OS are added to user space and do not require modification to kernel.**



- **Example: Mach (Maps UNIX system calls into appropriate user level services),**Taos, L4.
- Pros:
  - 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
- Cons:
  - Inefficient (boundary crossings)
  - Insufficient protection
  - Inconvenient to share data
  - between kernel and services

•A *virtual machine* takes the layered approach to its logical conclusion. It treats hardware and the operating system kernel as though they were all hardware.

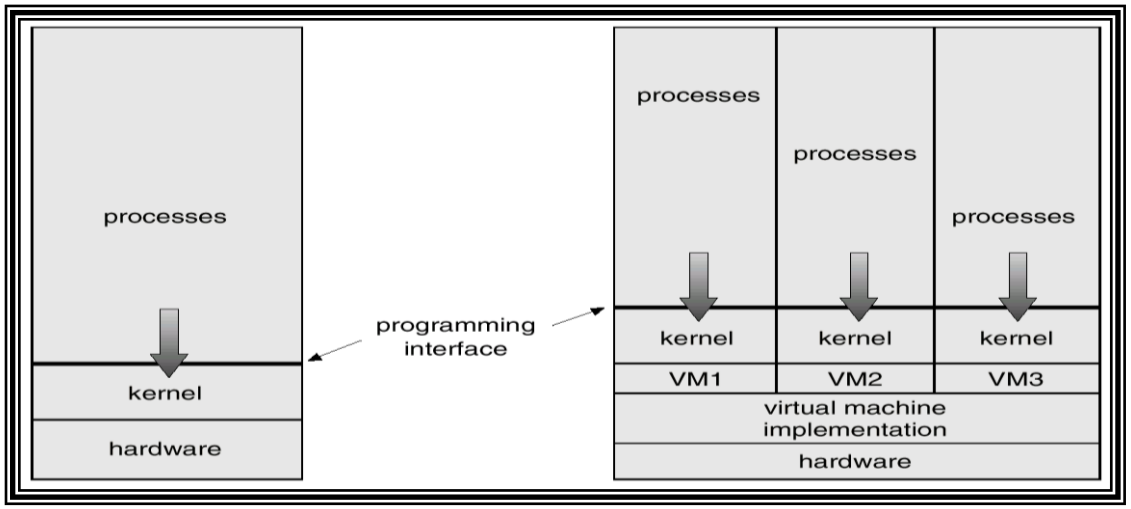
•A virtual machine provides an interface *identical* to the underlying bare hardware.

•The operating system creates the illusion of multiple processes, each executing on its own processor with its own (virtual) memory.

•The resources of the physical computer are shared to create the virtual machines.

- ✓ CPU scheduling can create the appearance that users have their own processor **- you get an "operators" console.**
- ✓ Spooling and a file system can provide virtual card readers and virtual line printers.
- ✓ A normal user time-sharing terminal serves as the virtual machine operator's console.

•Examples: IBM VM/370, Java VM, VMWare, Xen



Non-virtual Machine Virtual Machine

Shared resources: Mem, devices, disk, cpu, ….

Pros and Cons

- The virtual-machine concept provides complete protection of system resources since each virtual machine is isolated from all other virtual machines. This isolation, however, permits no direct sharing of resources **– shared only thru emulator**.
- A virtual-machine system is a perfect vehicle for operating-systems research and development. System development is done on the virtual machine, instead of on a physical machine and so does not disrupt normal system operation.
- The virtual machine concept is difficult to implement due to the effort required to provide an *exact* duplicate to the underlying machine.

Bibliography

- Silberschatz, A, Galvin, P.B, and Gagne, G., Operating System Principles, 9e, John Wiley & Sons, 2013.
- Stallings W., Operating Systems-Internals and Design Principles, 7e, Pearson Education, 2014.
- Harvey M. Deital, “Operating System”, Third Edition, Pearson Education, 2013.
- Andrew S. Tanenbaum, “Modern Operating Systems”, Second Edition, Pearson Education, 2004.
- Gary Nutt, “Operating Systems”, Third Edition, Pearson Education, 2004.

Acknowledgements

- I have drawn materials from various sources such as mentioned in bibliography or freely available on Internet to prepare this presentation.
- I sincerely acknowledge all sources, their contributions and extend my courtesy to use their contribution and knowledge for educational purpose.

Thank You!!

?