1.11 OPERATING SYSTEM STRUCTURE:

The operating systems are large and complex. A common approach is to partition the task into small components, or modules, rather than have one monolithic system.

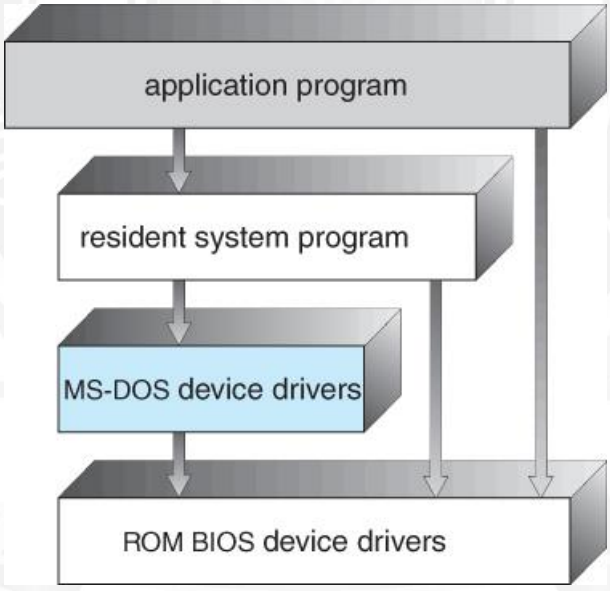
The structure of an operating system can be defined the following structures.

- Simple structure
- Layered approach
- Microkernels
- Modules
- Hybrid systems

Simple structure:

The Simple structured operating systems do not have a well-defined structure. These systems will be simple, small and limited systems.

Example: MS-DOS.

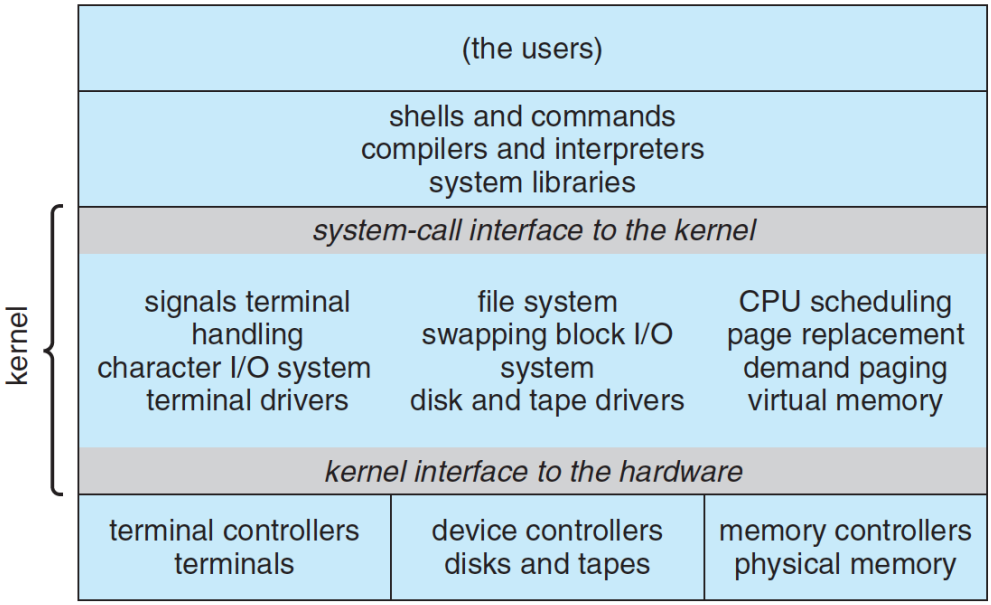


- In MS-DOS, the interfaces and levels of functionality are not well separated.
- In MS-DOS application programs are able to access the basic I/O routines. This causes the entire systems to be crashed when user programs fail.

Example: Traditional UNIX OS

It consists of two separable parts: the kernel and the system programs.

- The kernel is further separated into a series of interfaces and device drivers
- The kernel provides the file system, CPU scheduling, memory management, and other operating-system functions through system calls.



Layered approach:

- A system can be made modular in many ways. One method is the layered approach, in which the operating system is broken into a number of layers (levels). The bottom layer (layer 0) is the hardware; the highest layer 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 those data.
- The main advantage of the layered approach is simplicity of construction and debugging. The layers are selected so that each uses functions (operations) and services of only lower-level layers.
- Each layer is implemented only with 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.
- The major difficulty with the layered approach involves appropriately defining the various layers because a layer can use only lower-level layers.
- A problem with layered implementations is that they tend to be less efficient than other types.

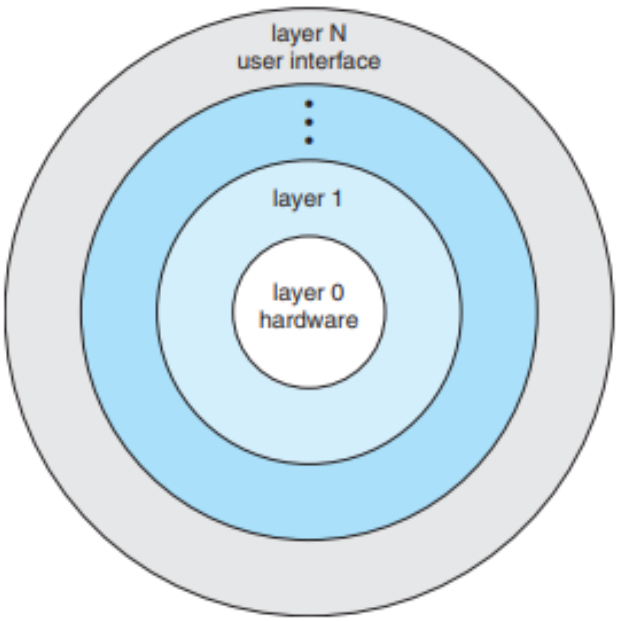


Fig : Layered Approach

Microkernels:

- In the mid-1980s, researchers at Carnegie Mellon University developed an operating system called Mach that modularized the kernel using the microkernel approach.
- This method structures the operating system by removing all nonessential components from the kernel and implementing them as system and user-level programs.
- Microkernel provide minimal process and memory management, in addition to a communication facility.
- The main function of the microkernel is to provide communication between the client program and the various services that are also running in user space.
- The client program and service never interact directly. Rather, they communicate indirectly by exchanging messages with the microkernel.
- One benefit of the microkernel approach is that it makes extending the operating system easier. All new services are added to user space and consequently do not require modification of the kernel.
- The performance of microkernel can suffer due to increased system-function overhead.

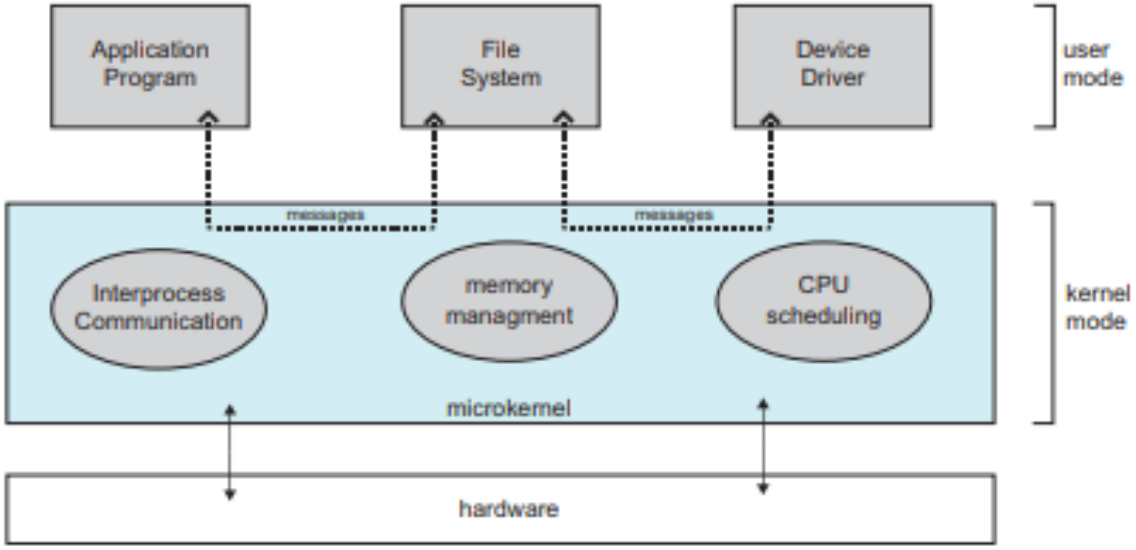


Fig : Architecture of typical microkernel

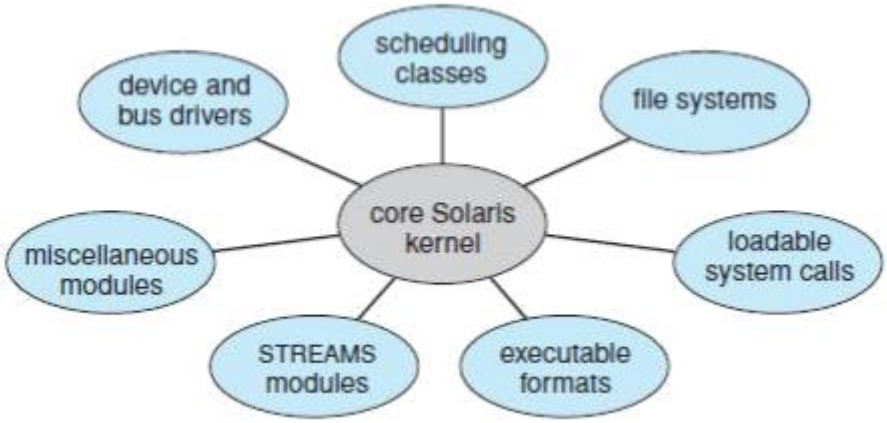
Modules:

- The best current methodology for operating-system design involves using loadable kernel modules
- The kernel has a set of core components and links in additional services via modules, either at boot time or during run time.
- The kernel provides core services while other services are implemented dynamically, as the kernel is running.
- Linking services dynamically is more comfortable than adding new features directly to the kernel, which would require recompiling the kernel every time a change was made.

Example: Solaris OS

The Solaris operating system structure is organized around a core kernel with seven types of loadable kernel modules:

- Scheduling classes
- File systems
- Loadable system calls
- Executable formats
- STREAMS modules
- Miscellaneous
- Device and bus drivers



Hybrid Systems:

- The Operating System combines different structures, resulting in hybrid systems that address performance, security, and usability issues.
- They are monolithic, because having the operating system in a single address space provides very efficient performance.
- However, they are also modular, so that new functionality can be dynamically added to the kernel.

Example: Linux and Solaris are monolithic (simple) and also modular, IOS.

1.12 OPERATING-SYSTEM OPERATIONS

Interrupt-driven nature of modern OS requires that erroneous processes not be able to disturb anything else.

Dual-Mode and Multimode Operation

- User mode when executing harmless code in user applications
- Kernel mode (a.k.a. system mode, supervisor mode, privileged mode) when executing potentially dangerous code in the system kernel.
- Certain machine instructions (privileged instructions) can only be executed in kernel mode.
- Kernel mode can only be entered by making system calls. User code cannot flip the mode switch.
- Modern computers support dual-mode operation in hardware, and therefore most modern OSes support dual-mode operation.

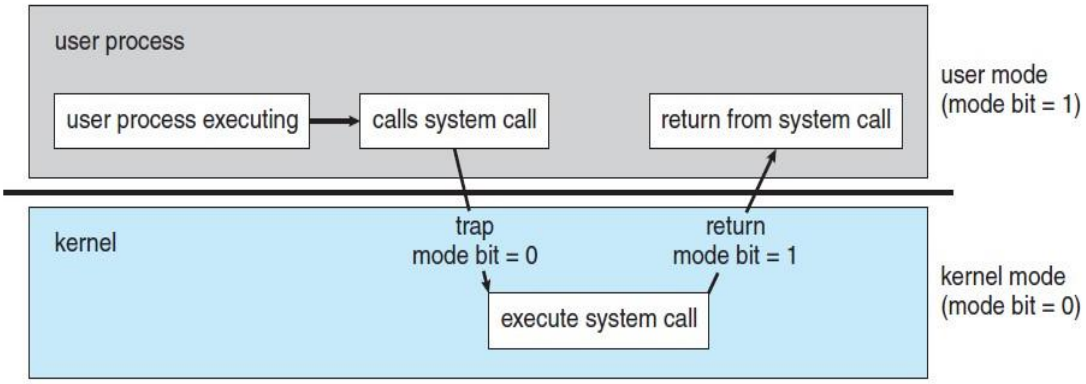


Fig : Transition from user to kernel mode

- The concept of modes can be extended beyond two, requiring more than a single mode bit
- CPUs that support virtualization use one of these extra bits to indicate when the virtual machine manager, VMM, is in control of the system. The VMM has more privileges than ordinary user programs, but not so many as the full kernel.
- System calls are typically implemented in the form of software interrupts, which causes the hardware's interrupt handler to transfer control over to an appropriate interrupt handler, which is part of the operating system, switching the mode bit to kernel mode in the process.
- The interrupt handler checks exactly which interrupt was generated, checks additional parameters (generally passed through registers) if appropriate, and then calls the appropriate kernel service routine to handle the service requested by the system call.
- User programs' attempts to execute illegal instructions (privileged or non- existent instructions), or to access forbidden memory areas, also generate software interrupts, which are trapped by the interrupt handler and control is transferred to the OS, which issues an appropriate error message, possibly dumps data to a log (core) file for later analysis, and then terminates the offending program.

Timer

- Before the kernel begins executing user code, a timer is set to generate an interrupt.
- The timer interrupt handler reverts control back to the kernel.
- This assures that no user process can take over the system.
- Timer control is a privileged instruction, (requiring kernel mode.)