**OPERATING SYSTEM STRUCTURE**

An operating system is a construct that allows the user application programs to interact with the system hardware. A modern operating system is large and complex, if it is to be function properly and be modified easily, then it must be engineered very carefully. A common approach is to partition the task into small components, or modules, rather than having one **monolithic**system. Each of these modules should be a well-defined with inputs, outputs, and functions.

Followings are different types of operating system structure.

1. Simple Structure
2. Layered Approach
3. Microkernels
4. Modules
5. Hybrid Systems

**1)Simple Structure:** This was the initially designed structure of the [operating system](https://er.yuvayana.org/operating-system-introduction-and-different-views-of-system/) and it was not a well-defined structure. Such systems started as small, simple, and then grew beyond their original scope. Two examples of Simple Structured Operating system are

* MS-DOS
* UNIX

MS-DOS was originally designed and implemented by a few people who had no idea that it would become so popular. It was written to provide the most functionality in the least space, so it was not carefully divided into modules. In MS-DOS, the interfaces and levels of functionality are not well separated. For instance, application programs are able to access the basic I/O routines to write directly to the display and disk drives. Such freedom leaves MS-DOS vulnerable to errant (or malicious) programs, causing entire system crashes when user programs fail.

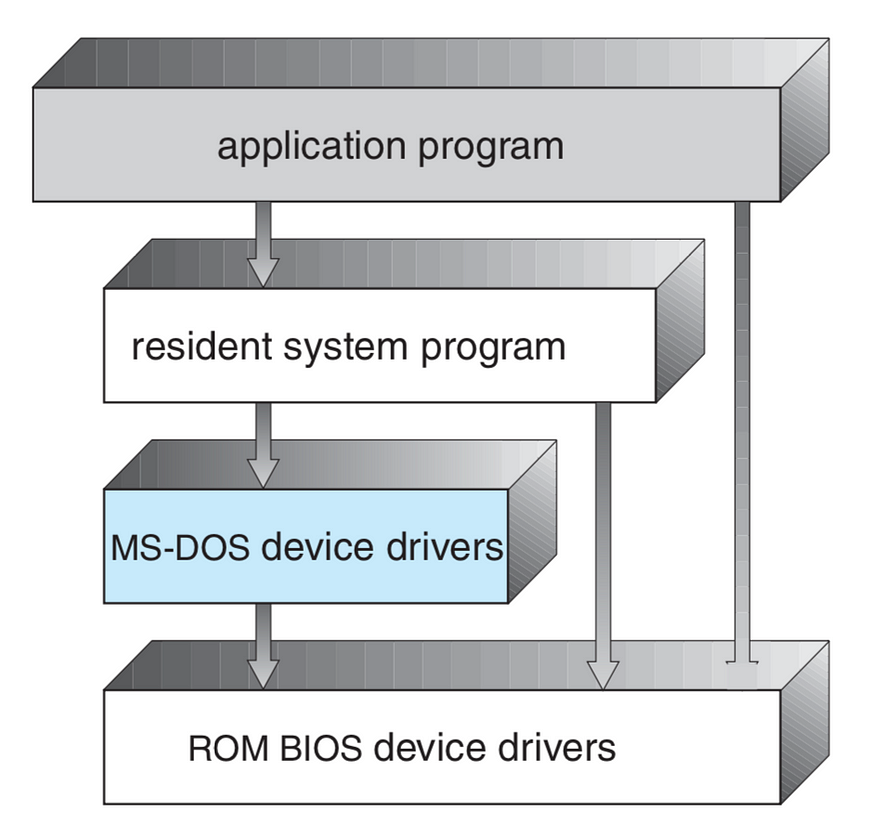
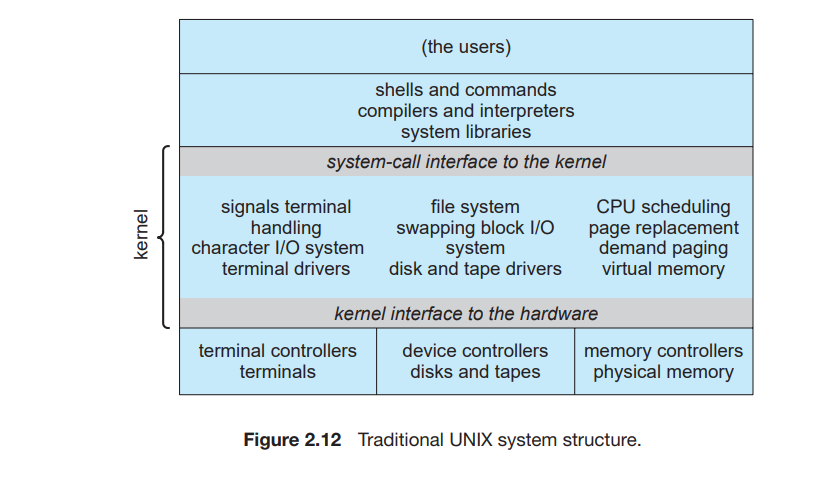


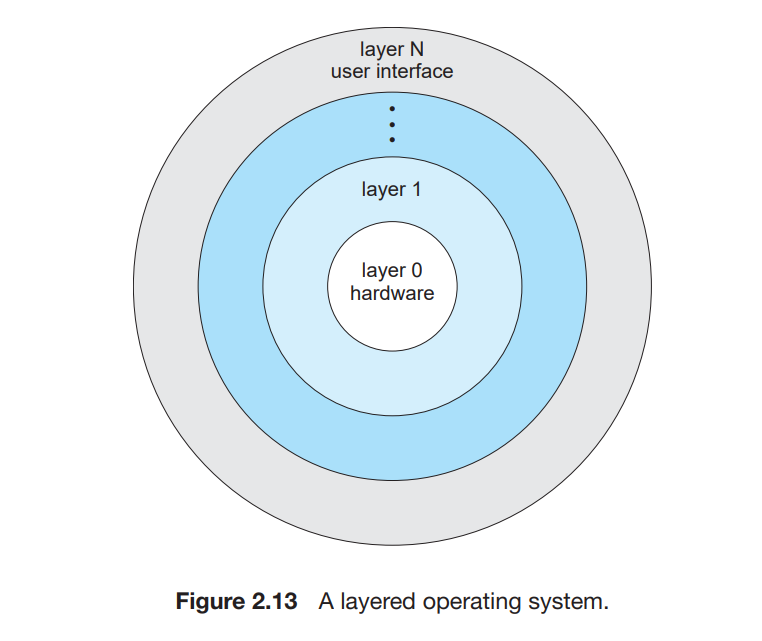
Fig: MS-DOS Layer Structure

Intel 8088 for which it was written provides no dual mode and no hardware protection, the designers of MS-DOS had no choice but to leave the base hardware accessible.

Another example of limited structuring is the original UNIX operating system. Like MS-DOS, UNIX initially was limited by hardware functionality. 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, which have been added and expanded over the years as UNIX has evolved. The kernel provides the file system, CPU scheduling, memory management, and other operating-system functions through system calls. Taken in sum, that is an enormous amount of functionality to be combined into one level. This monolithic structure was difficult to implement and maintain. There is very little overhead in the system call or in communication within the kernel.

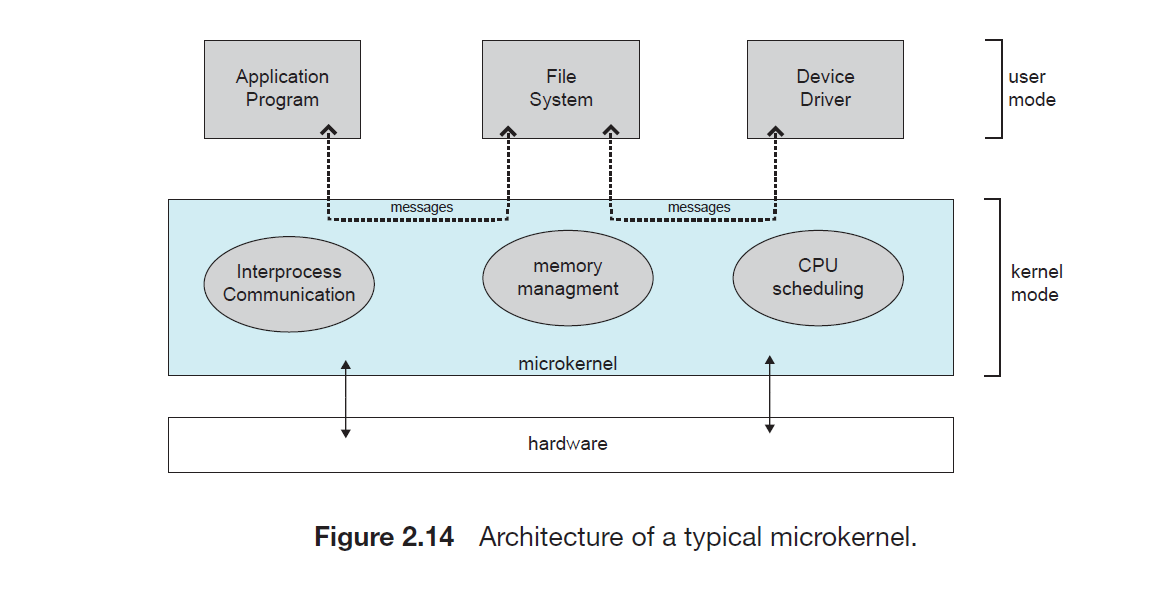
# 2) Layered Approach: A system can be made modular in many ways. One method is the layered approach, in which the operating system is broken up into a number of layers (levels). The bottom layer (layer 0) is the hardware; the highest (layer N) is the user interface.



The main advantage of the layered approach is simplicity of construction and debugging. The layers are selected so that each uses functions and services of only lower-level layers. That means, As a level is only using the functions of the lower level and upper levels has no effect on it, this makes the debugging easy. Level 0 can be easily debugged because it only involves hardware debugging. Similarly, level 1 can also be easily debugged because as we already checked level 0 has no errors, now we need to only check for level 1 and so other levels can also be easily debugged.

But this approach also involves some disadvantages. The first one is that designing is a bit complex. Because careful planning is required for designing as each level should only use the functionalities of its lower level. Another disadvantage is that if a system call is made for the lower level then the system call is passed through layers in a top-down approach.

**3)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. The result is a smaller kernel. This small kernel is called a microkernel.

There is little difficulty here which services should remain in the kernel and which should be implemented in user space. 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. Communication is provided through message passing.

**Ex:**  If the client program wishes to access a file, it must interact with the file server. 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 microkernel also provides more security and reliability, since most services are running as user—rather than kernel—processes. If a service fails, the rest of the operating system remains untouched.

But still, this model had a disadvantage in that the performance of Microkernels can suffer due to increased system function overhead.

**4)Modules:**

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

The overall result resembles a layered system in that each kernel section has defined, protected interfaces; but it is more flexible than a layered system, because any module can call any other module. The approach is also similar to the microkernel approach in that the primary module has only core functions and knowledge of how to load and communicate with other modules; but it is more efficient, because modules do not need to invoke message passing in order to communicate.

The Solaris operating system structure, shown in diagram, is organised around a core kernel with seven types of loadable kernel modules:

1. Scheduling classes

2. File systems

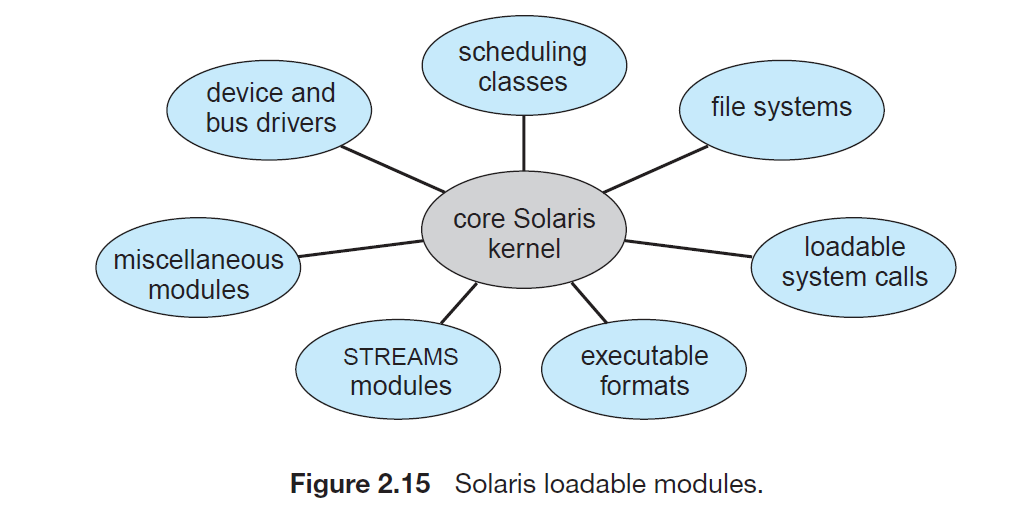
3. Loadable system calls

4. Executable formats

5. STREAMS modules

6. Miscellaneous

7. Device and bus drivers



**5)Hybrid Systems:**

In practice, very few operating systems adopt a single, strictly defined structure. Instead, they combine different structures, resulting in hybrid systems that address performance, security, and usability issues.

Three hybrid systems are well-known: the Apple Mac OS X operating system and the two most prominent mobile operating systems — iOS and Android.

