OPERATING SYSTEM STRUCTURE

Modern Operating Systems are large and complex and are carefully engineered to function properly and be modified easily. A common approach is to partition the task into small components. Each component is well-defined with carefully defined inputs, outputs, and function. The components are then interconnected and melded into a kernel.

OS Structure - types

- Simple Structure
- Layered Approach
- Microkernels Approach
- Modules
- Hybrid Systems

Simple Structure

Initially OS were small, simple and had limited functionality. They were designed and implemented to provide the most functionality in the least space and were not divided into modules. Examples of OS with simple monolithic structures are MS-DOS and UNIX. Disadvantage - Too much functionality combined into one level and hence difficult to implement, maintain and enhance

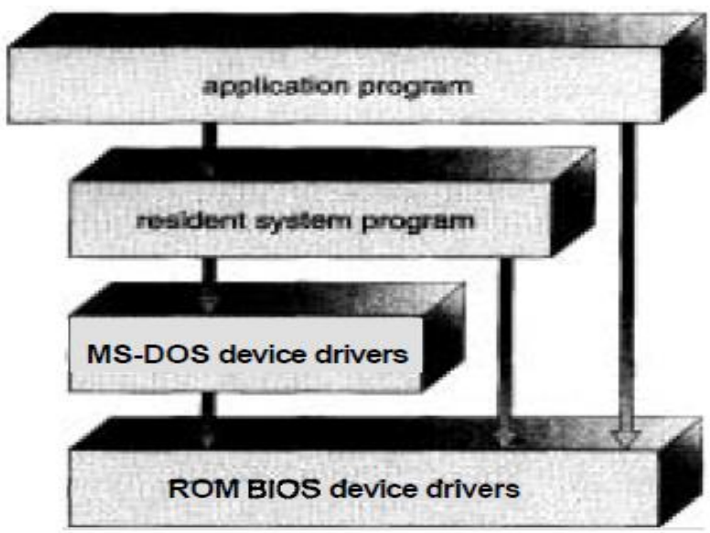

Figure 3.6 MS-DOS layer structure.

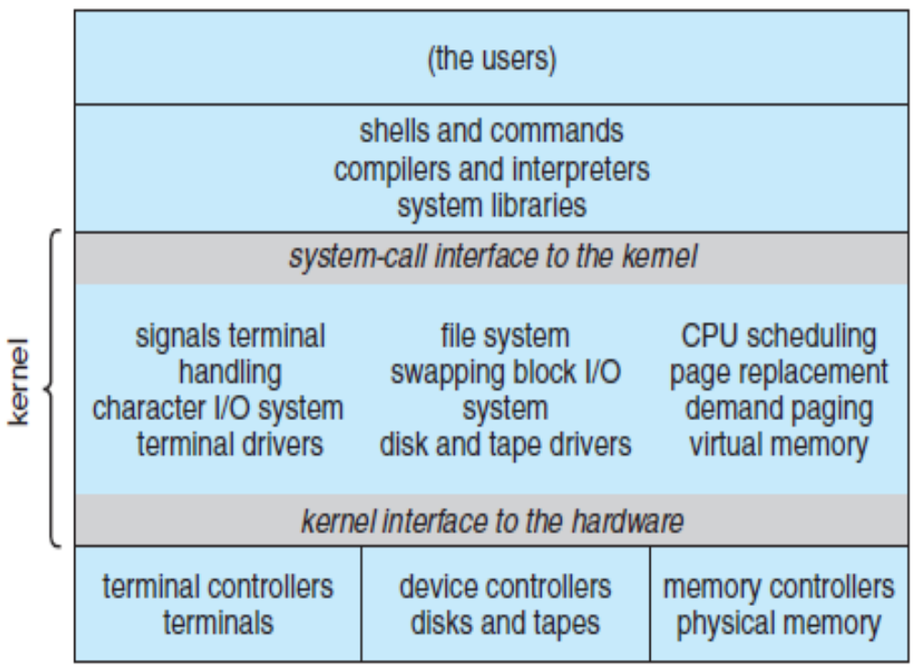

Figure 2.12 Traditional UNIX system structure.

Layered Approach

In layered approach, the overall functionality and features of the OS are determined and are separated into smaller, more appropriate components. Related components are then grouped and placed in a layer. The bottom layer (layer 0) is the hardware; and the highest (layer N) is the user interface. The layers are selected such that each uses functions and services of its lower-level layers. Also the layer needs to know only what operations are done at each layer not how these operations are implemented. This allows the layer to hide the existence of its data structures, operations and hardware from higher-level layers (abstraction). This approach also simplifies debugging and system verification.

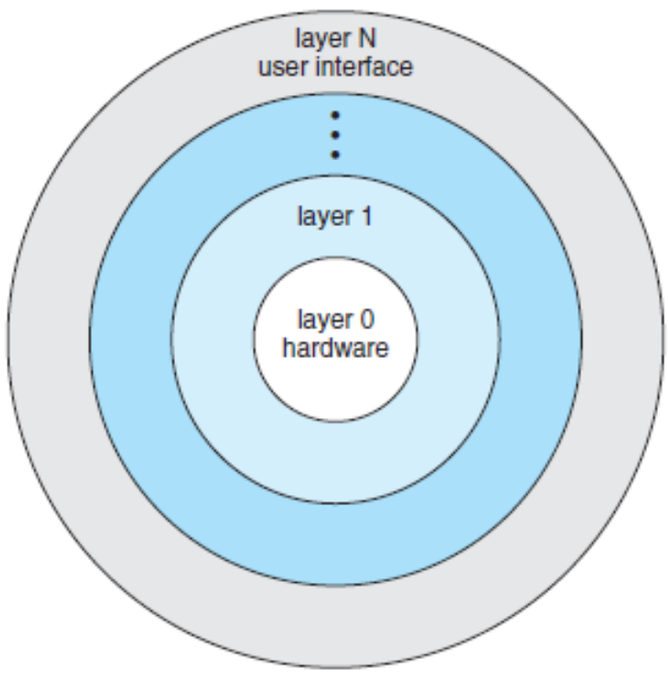

Figure 2.13 A layered operating system.

- Disadvantages
    1. Needs careful definition of the layers, because a layer can use only those layers below it.
    2. They tend to be less efficient than other types.

Solution is to use fewer layers with more functionality to provide the advantages of modularized code while avoiding the difficult problems of layer definition and interaction.

Microkernels Approach

In Microkernel approach, the kernel is modularized and structured by removing all nonessential components from the kernel, and implementing them as system and user level programs resulting in a smaller kernel. Microkernels typically provide process management and memory management and communication facility. Communication between client program and services running in user space is provided by exchanging messages with the microkernel. Architecture of a typical microkernel is as below

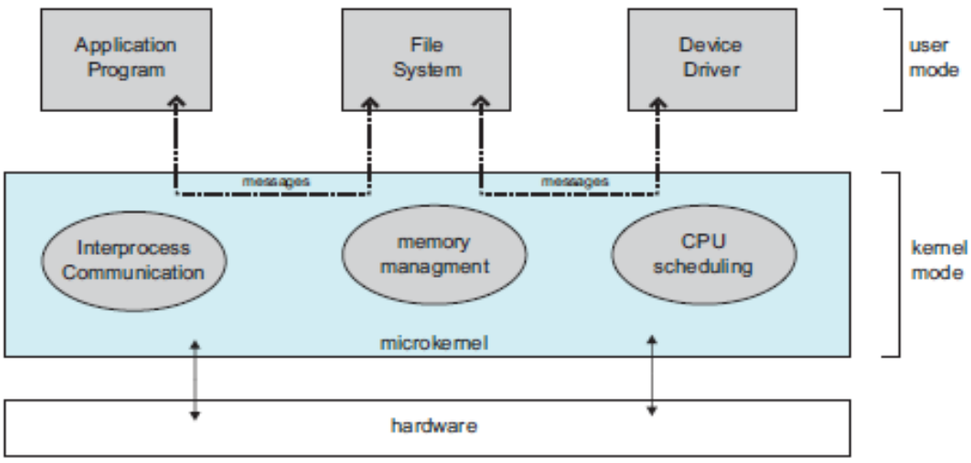

Figure 2.14 Architecture of a typical microkernel.

The benefits of the microkernel approach

1. Easy to extend the OS – All new services are added to user space and consequently do not require modification of the kernel.
2. When the kernel has to be modified, the changes tend to be fewer, because the microkernel is a smaller kernel.
3. The resulting operating system is easier to port from one hardware design to another.
4. The microkernel also provides more security and reliability, since most services are running as user rather than kernel processes.
5. If a service fails, the rest of the operating system remains untouched.

- Disadvantage of microkernel approach – increased system function overhead

Loadable Kernel Modules

In this approach the Kernel has a set of core components and dynamically links in additional services via modules, either at boot time or during run time. Linking services dynamically is easier than adding new features directly to the kernel (will require recompiling the kernel every time a change is made). This type of design is common in modern implementations of UNIX, such as Solaris, Linux, and Mac OS X, as well as Windows.

Advantages

1. Each kernel section has defined, protected interfaces;
2. Flexible – any module can call any other module.
3. Efficient – modules do not need to invoke message passing in order to communicate.

Example – Solaris OS, Linux. Solaris OS is organized around a core kernel with seven types of loadable kernel modules as shown below

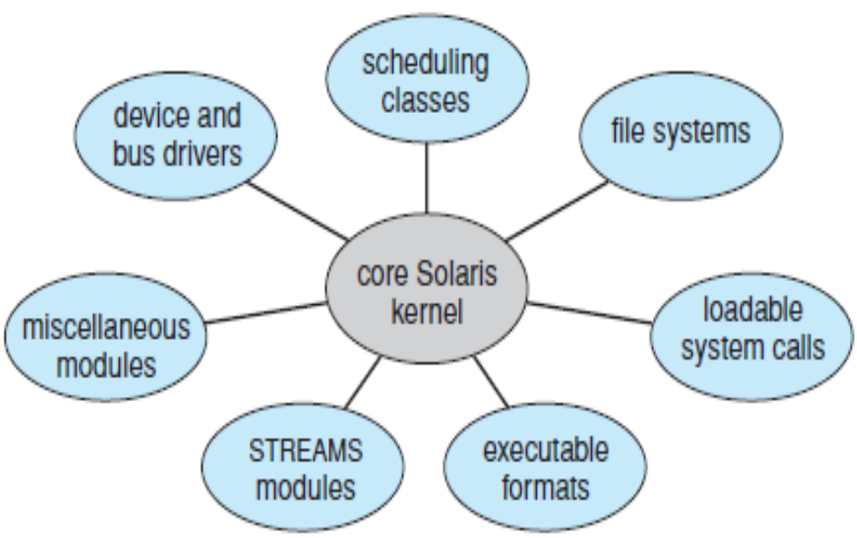

Figure 2.15 Solaris loadable modules.

Hybrid Systems

In practice, OS combine different structures, resulting in hybrid systems that address performance, security and usability issues.

Example

1. Linux and Solaris use monolithic structure, (having the OS in a single address space provides very efficient performance). They are also modular, (new functionality can be dynamically added to the kernel).
2. Windows – monolithic – but it retains some behaviour typical of microkernel systems, and provides support for dynamically loadable kernel modules.

Three other popular hybrid systems:

1. Apple Mac OS X operating system and
2. Mobile operating systems – iOS and Android.

Android

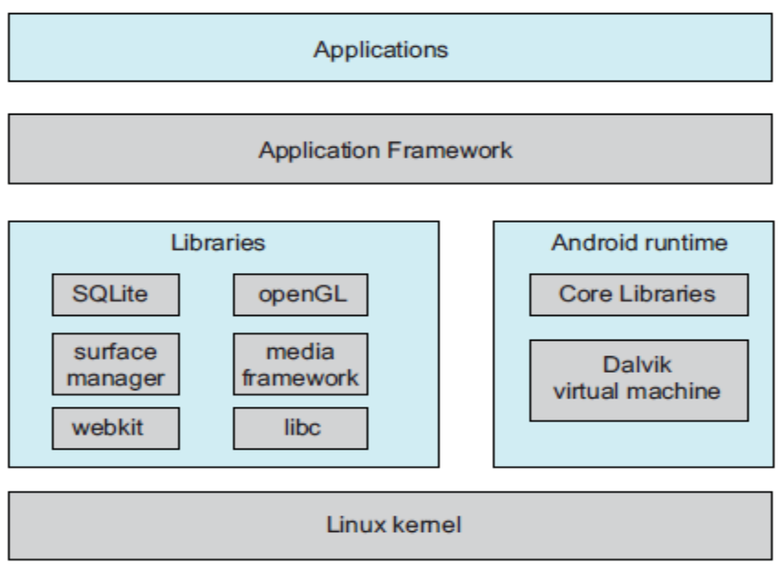

Figure 2.18 Architecture of Google's Android.

Software stack

- The Linux kernel – supports process, memory, device and power management
- The Android runtime environment includes a core set of libraries and the Dalvik virtual machine.
- Other libraries includes frameworks for developing web browsers (webkit), database support (SQLite), and multimedia.

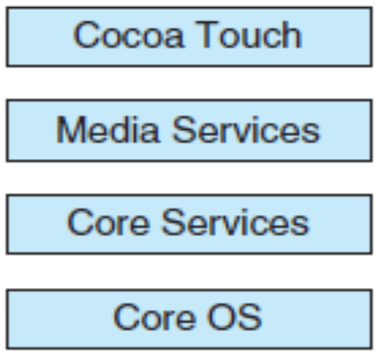

Figure 2.17 Architecture of Apple's iOS.

Software stack

- Core OS – based on the kernel environment
- Core services – new features – cloud computing, databases
- Media service layer – for graphics, audio, and video
- Cocoa Touch – API – provides frameworks for developing applications that run on iOS devices. Eg. API for Touch screens