**Operating Systems**

Lecture Notes – Week 9

*G.  MacKenzie, I. Bojanova, Z. Xu*

**Introduction**

Our primary topics this week include:

* Role of Operating System software in a computer system
* Operating Systems concept and components
* Operating System services
* Review of different types of Operating Systems
* Review of different types of user interfaces

This lecture adds a few thoughts to augment the Englander material.

Acknowledgement: the lecture was derived, in part, from the notes of Dr. David Madison and from Tanenbaum (2001).

**History of Operating Systems**

The operating system (OS) serves as the interface between the computer's hardware and the application or "outside world." Operating systems have historically been closely tied to the computers on which they run. They have evolved along with the generations of computers that we reviewed in an earlier session, as shown below:

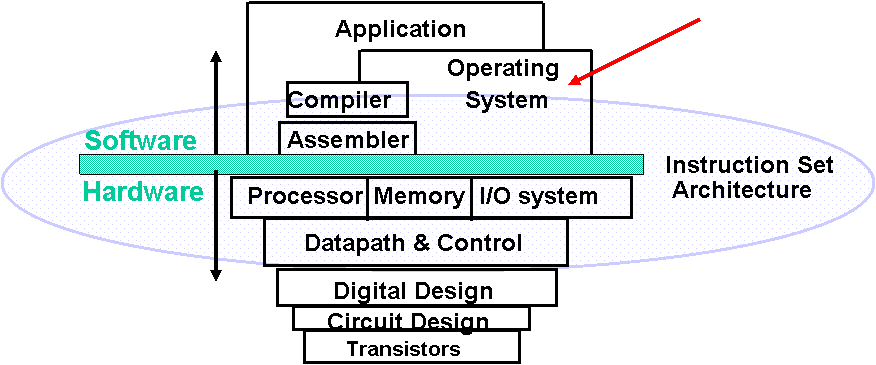
* First Generation (1946 - 1957): Vacuum Tubes and Plugboards
  + Programming languages and operating systems were unknown
  + Programming was done by wiring plugboards
* Second Generation (1958 - 1964): Transistors
  + Initially, punch-card jobs were run individually. Eventually, program decks were stacked and run as batch jobs
* Representative OSs: FMS (Fortran Monitor System), IBSYS (IBM OS for the 7094)  
  Third Generation (1965 - 1971): Integrated Circuits
  + IBM System/360 introduced the concept of computer “families,” necessitating much more complicated Oss
  + IBM developed OS/360, capable of multiprogramming

Jobs were queued on disk; technique known as “spooling” (Simultaneous Peripheral Operation On Line)

* + Timesharing OSs were developed. Bell Labs, GE, and MIT teamed to build MULTICS (MULTiplexed Information and Computing Service), designed to handle 100s of simultaneous users
  + A slimmed-down version of MULTICS, UNIX, was developed by Ken Thompson of Bell Labs
* Fourth Generation and beyond (1972 -present): Personal Computers
  + MS-DOS, with many features of UNIX, developed by Microsoft
  + UNIX became popular with powerful workstations
  + Network and distributed OSs became available for PCs
  + PC OSs became GUI (Graphical User Interface) based, e.g., Windows, MAC O/S, etc.

**Operating Systems Concepts**

We’ve seen from our earlier look at computer architectures  (instruction set, memory organization, I/O, and bus structure) that programming at the machine level is primitive and awkward, so one solution is to present the user with the equivalent of an *extended machine* or a *virtual machine* that is easier to program than the underlying hardware. The OS is responsible for task control, memory management, and I/O handling. Fig. 9.1 shows the Operating System as one component of a computer structure consisting of various physical and logical layers.



***Figure 9.1 - "Layers" of a Computer***

The previous view of operating systems is top-down: it presents the OS as a convenient machine interface. An alternative, bottom-up view presents the OS as a resource manager to schedule jobs, insure security, manage memory, keep I/O sorted, mitigate conflicting requests, etc. In reality, one would like the operating system to accomplish both of these functions, i.e., present a simple interface (extended   machine), and manage the computer’s resources. The interface between an OS and the user/programs is via system calls that fall into two broad categories: those that deal with the creation/termination of processes, and those that deal with files.

A key concept in all operating systems is the *process*, which is a program in execution. It consists of an executable program, the program’s data and stack, and program counter, stack pointer, and register information. Modern operating systems have the ability to start, suspend, and resume processes, and note that the operating system is itself a process.

*Files* are a mechanism for hiding peculiarities of disks and other I/O devices. Most OSs organize files into *directories*.Both processes and files are organized into hierarchical “trees,” but file hierarchies are generally deeper. At any instance, a process has a current working directory, and OSs provide means for providing file protection, i.e., access control. Many OSs treat file and I/O access similarly.

*System calls* can be made from software or from a *shell*, which provides an interface between a user and an operating system. The shell, itself a process, interprets user command and makes system calls.

**Operating System Structures**

Operating systems generally fall into one of the four categories listed below and illustrated in Figures 9.2 - 9.5:

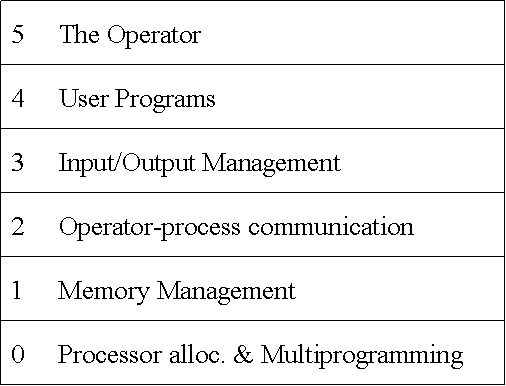
* Monolithic Systems (Fig. 9.2)
  + Operating system consists of a collection of procedures, with little layering or structure
* Layered Systems (Fig. 9.3)
  + OS organized as a hierarchy of layers, with each layer communicating with its neighbors

MULTICS, later UNIX, uses variation of layers

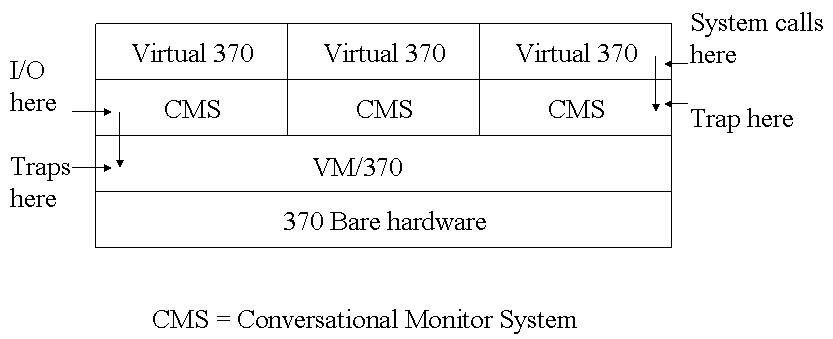
* Virtual Machines (Fig. 9.4)
  + Multiple copies of the machine are simulated as a virtual machine
  + Each virtual machine can run an arbitrary OS
  + Example system: IBMs VM/370
* Client-Server Models (Fig. 9.5)
  + Minimize the OS into a kernel
  + Requests for service are generated by a client process, which is then passed through the kernal to a server process
  + The location of the server process can be transparent to a user
  + Client-Server Models form the basis of modern distributed systems

****

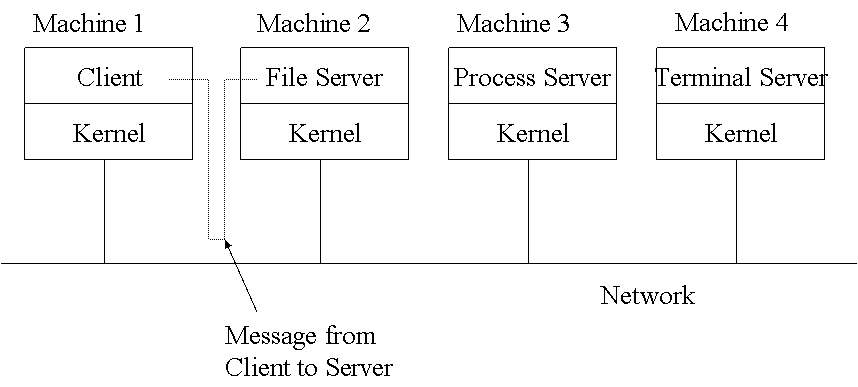
***Figure 9.2 - Monolithic System***



***Figure 9.3 - Layered System***



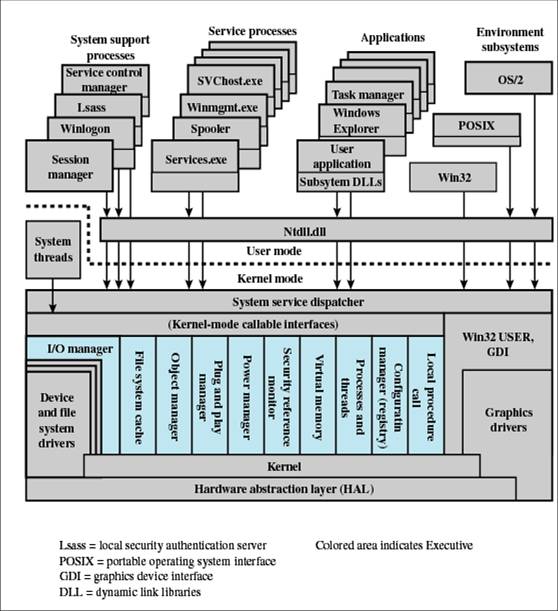
***Figure 9.4 - Virtual Machine***



***Figure 9.5 - Client-Server Model***

**Operating System Example**

In case the above figures appear overly simplistic, and you think the operating systems should be relatively simple pieces of software, please look at the basic architecture of Windows 2000 shown in Fig. 9.6. A successor to "2000," MS Vista, supposedly contains about 50 million lines of code - easily two orders of magnitude more than the early versions of MS DOS.

***  
  
Figure 9.6 - Windows 2000 Architecture***

**User Interface**

* Command Line
  + Single line
  + Operands
  + Switches/modifiers
  + Batch
  + Script
* Graphical User Interface
  + Keyboard/Mouse
  + Touch Screen
  + Gesture/Voice
* User Function and Program Services

**Summary**

As of a few years ago it was relatively easy to classify operating systems in another way. There were proprietary OSs that were vendor dependent, such as Windows, IBM MVS, DEC VMS, and Macintosh OS-X. And there were "open" UNIX-based OSs like IBM AIX, Sun Solaris, HP UNIX, and Linux. Today, with the proliferation of both mobile devices driven in part by specialized operating systems derived largely from freely available open source software, the landscape has become much more confusing.

Here is a comparison among Windows, Mac and Linux operating systems: <https://www.slideshare.net/MeharAliZar/comparison-of-windows-linux-and-mac-os>

References

Tanenbaum, A. S. (2001), *Modern operating systems (2nd ed.)*, Prentice Hall.