Department of Electronics & Communication Engg.

Course : Operating Systems -15EC553 **Sem.: 5th (2018-19 ODD)**

Course Coordinator:

Prof. Nyamatulla M Patel

Operating Systems-15EC553

Module-02
Structure of Operating System

Introduction

- Operation of an OS
- Structure of an Operating System
- Operating Systems with Monolithic Structure
- Layered Design of Operating Systems

Introduction (continued)

- Virtual Machine Operating Systems
- Kernel-Based Operating Systems
- Microkernel-Based Operating Systems
- Case Studies

Operation of an OS

- When a computer is switched on, *boot procedure*
  - analyzes its configuration—CPU type, memory size, I/O devices, and details of other hardware
  - Loads part of OS in memory, initializes data structures, and hands it control of computer system
- During operation of the computer, interrupts caused by:
  - An event: I/O completion; end of a time slice
  - *System call* made by a process *(software interrupt)*
- Interrupt servicing routine:
  - Performs *context save*
  - Activates *event handler*
- Scheduler selects a process for servicing

Operation of an OS (continued)

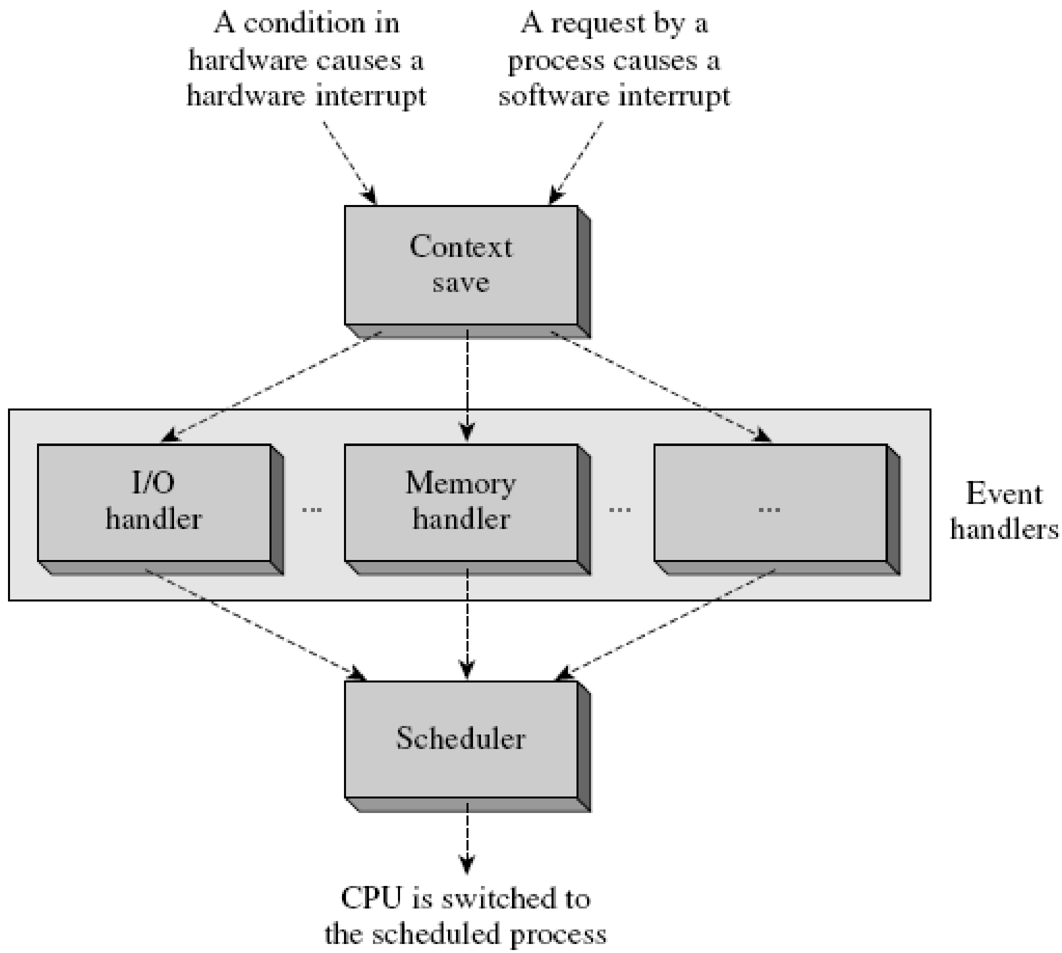

Figure 4.1 Overview of OS operation.

Operation of an OS (continued)

Table 4.1 Functions of an OS

Function	Description
Process management	Initiation and termination of processes, scheduling
Memory management	Allocation and deallocation of memory, swapping, virtual memory management
I/O management	I/O interrupt servicing, initiation of I/O operations, optimization of I/O device performance
File management	Creation, storage and access of files
Security and protection	Preventing interference with processes and resources
Network management	Sending and receiving of data over the network

Structure of an Operating System

- Policies and Mechanisms
- Portability and Extensibility of Operating Systems

Policies and Mechanisms

- In determining how OS will perform a function, designer needs to think at two distinct levels:
  - *Policy:* Principle under which OS performs function
    - Decides *what* should be done
  - *Mechanism:* Action needed to implement a policy
    - Determines *how* to do it and actually does it
- Example:
  - Round-robin scheduling is a policy
  - Mechanisms: maintain queue of ready processes and *dispatch* a process

Portability and Extensibility of Operating Systems

- *Porting:* adapting software for use in a new computer system
- *Portability:* ease with which a software program can be ported
  - Inversely proportional to the porting effort
- Porting an OS: changing parts of its code that are architecture-dependent to work with new HW
  - Examples of architecture-dependent data and instructions in an OS:
    - Interrupt vector, memory protection information, I/O instructions, etc.

Portability and Extensibility of Operating Systems (continued)

- *Extensibility:* ease with which new functionalities can be added to a software system
  - Extensibility of an OS is needed for two purposes:
    - Incorporating new HW in a computer system
      - Typically new I/O devices or network adapters
    - Providing new features for new user expectations
  - Early OSs did not provide either kind of extensibility
  - Modern OSs facilitate addition of a *device driver*
    - They also provide a *plug-and-play* capability

Operating Systems with Monolithic Structure

- Early OSs had a *monolithic* structure
  - OS formed a single software layer between the user and the bare machine (hardware)

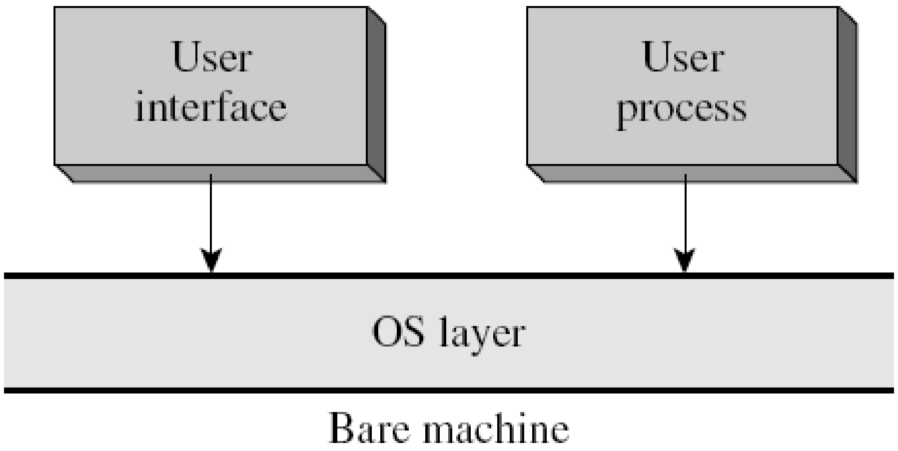

Figure 4.2 Monolithic OS.

Operating Systems with Monolithic Structure (continued)

- Problems with the monolithic structure:
  - Sole OS layer had an interface with bare machine
    - Architecture-dependent code spread throughout OS
      - Poor portability
    - Made testing and debugging difficult
      - High costs of maintenance and enhancement
- Alternative ways to structure an OS:
  - *Layered structure*
  - *Kernel-based structure*
  - *Microkernel-based OS structure*

Layered Design of Operating Systems

Definition 4.1 Semantic Gap The mismatch between the nature of operations needed in the application and the nature of operations provided in the machine.

- Semantic gap is reduced by:
  - Using a more capable machine
  - Simulating an *extended machine* in a lower layer

Layered Design of Operating Systems (continued)

- Routines of one layer must use only the facilities of the layer directly below it
  - Through its interfaces only

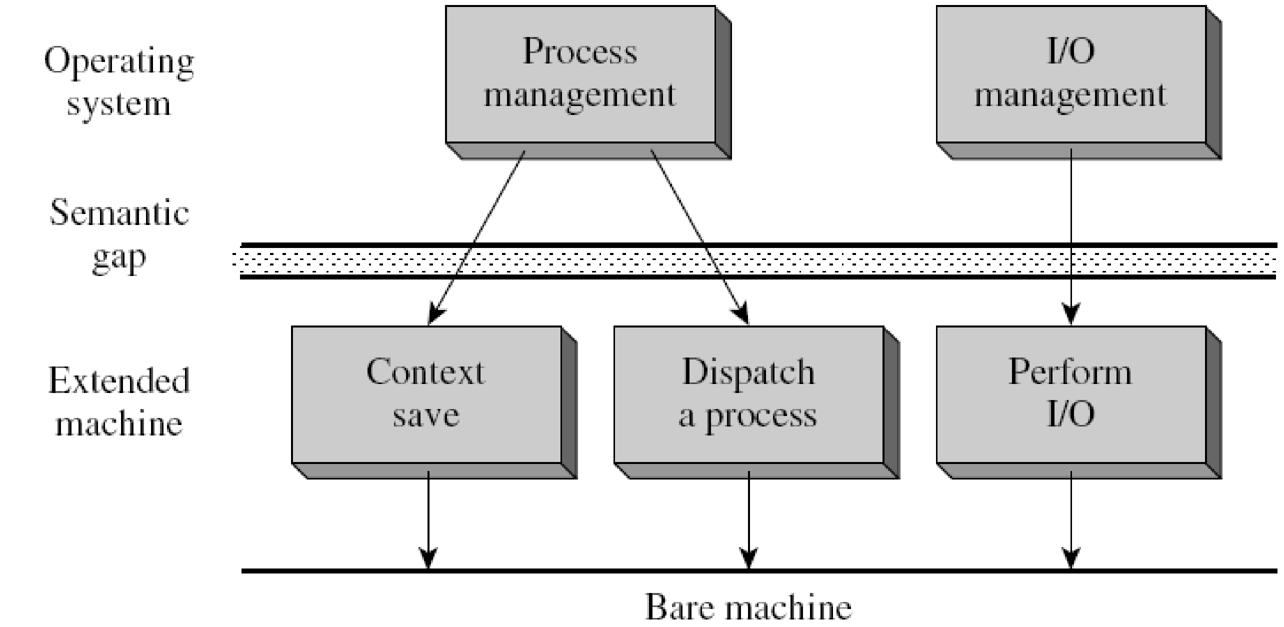

Figure 4.4 Layered OS design.

Example: Structure of the THE Multiprogramming System

Table 4.2 Layers in the THE Multiprogramming System

Layer	Description
Layer 0	Processor allocation and multiprogramming
Layer 1	Memory and drum management
Layer 2	Operator–process communication
Layer 3	I/O management
Layer 4	User processes

Layered Design of Operating Systems (continued)

- Problems:
  - System operation slowed down by layered structure
  - Difficulties in developing a layered design
    - Problem: ordering of layers that require each other's services
      - Often solved by splitting a layer into two and putting other layers between the two halves
  - Stratification of OS functionalities
    - Complex design
    - Loss of execution efficiency
    - Poor extensibility

Virtual Machine Operating Systems

- Different classes of users need different kinds of user service
- *Virtual machine operating system* (VM OS) creates several *virtual machines*
  - A *virtual machine* (VM) is a virtual resource
  - Each VM is allocated to one user, who can use any OS
    - *Guest OSs* run on each VM
- VM OS runs in the real machine
  - schedules between guest OSs
- Distinction between privileged and user modes of CPU causes some difficulties in use of a VM OS

Example: Structure of VM/370

Figure 4.5 Virtual machine operating system VM/370.

Virtual Machine Operating Systems (continued)

- *Virtualization:* mapping interfaces and resources of a VM into interfaces and resources of host machine
  - Full virtualization may weaken security
  - *Paravirtualization* replaces a nonvirtualizable instruction by easily virtualized instructions
    - Code of a guest OS is modified to avoid use of nonvirtualizable instructions, done by:
      - *Porting* guest OS to operate under VM OS
      - Or, using *dynamic binary translation* for kernel of a guest OS

Virtual Machine Operating Systems (continued)

- VMs are employed for diverse purposes:
  - *Workload consolidation*
  - To provide security and reliability for applications that use the same host and the same OS
  - To test a modified OS on a server concurrently with production runs of that OS
  - To provide disaster management capabilities
    - A VM is transferred from a server that has to shutdown to another server available on the network

Virtual Machine Operating Systems (continued)

- VMs are also used without a VM OS:
  - Virtual Machine Monitor (VMM)
    - Also called a *hypervisor*
    - E.g., VMware and XEN
  - Programming Language Virtual Machines
    - Pascal in the 70s
      - Substantial performance penalty
    - Java
      - Java virtual machine (JVM) for security and reliability
      - Performance penalty can be offset by implementing JVM in hardware

Kernel-Based Operating Systems

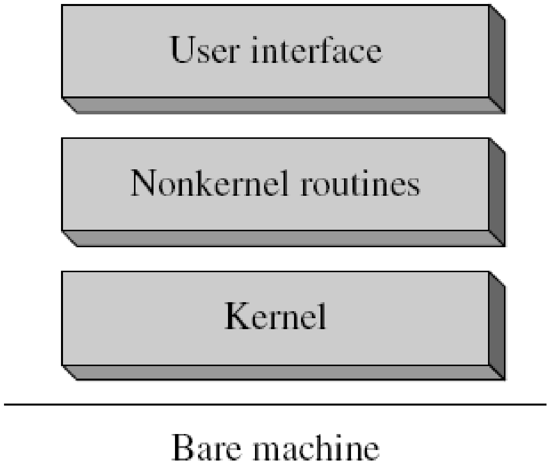

Figure 4.6 Structure of a kernel-based OS.

- Historical motivations for kernel-based OS structure were OS portability and convenience in design and coding of nonkernel routines
  - *Mechanisms* implemented in kernel, *policies* outside
- Kernel-based OSs have poor extensibility

Kernel-Based Operating Systems (continued)

Table 4.3 Typical Functions and Services Offered by the Kernel

OS functionality	Examples of kernel functions and services
Process management	Save context of the interrupted program, dispatch a process, manipulate scheduling lists
Process communication	Send and receive interprocess messages
Memory management	Set memory protection information, swap-in/swap-out, handle page fault (that is, "missing from memory" interrupt of Section 1.4)
I/O management	Initiate I/O, process I/O completion interrupt, recover from I/O errors
File management	Open a file, read/write data
Security and protection	Add authentication information for a new user, maintain information for file protection
Network management	Send/receive data through a message

Evolution of Kernel-Based Structure of Operating Systems

- *Dynamically loadable kernel modules*
  - Kernel designed as set of modules
    - Modules interact through interfaces
  - *Base kernel* loaded when system is booted
    - Other modules loaded when needed
      - Conserves memory
  - Used to implement device drivers and new system calls
- *User-level device drivers*
  - Ease of development, debugging, deployment and robustness
  - Performance is ensured through HW and SW means

Microkernel-Based Operating Systems

- The *microkernel* was developed in the early 1990s to overcome the problems concerning portability, extensibility, and reliability of kernels
- A microkernel is an essential core of OS code
  - Contains only a subset of the mechanisms typically included in a kernel
  - Supports only a small number of system calls, which are heavily tested and used
  - Less essential code exists outside the kernel

Microkernel-Based Operating Systems (continued)

- Microkernel does not include scheduler and memory handler
- They execute as servers

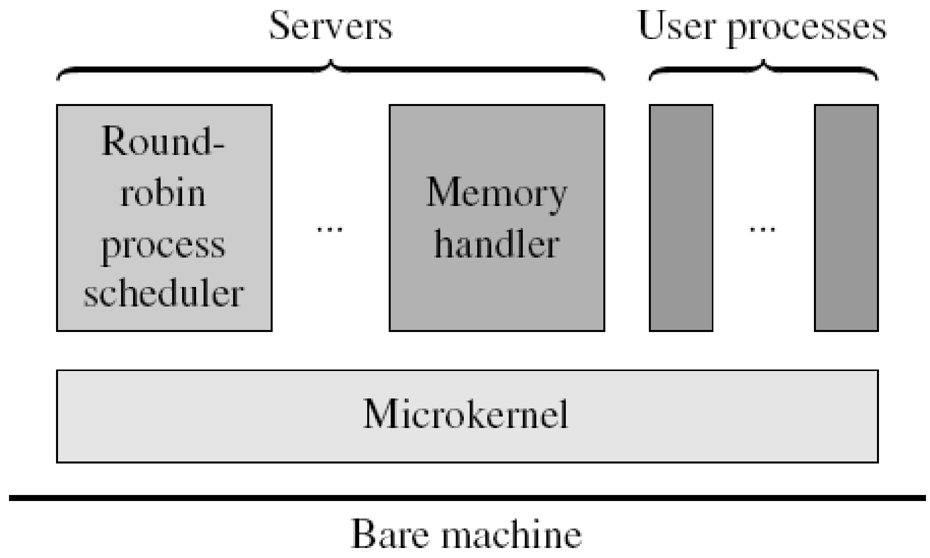

Figure 4.7 Structure of microkernel-based operating systems.

Microkernel-Based Operating Systems (continued)

- Considerable variation exists in the services included in a microkernel
- OSs using first-generation microkernels suffered up to 50% degradation in throughput
  - L4 microkernel is second-generation
    - Made IPC more efficient by eliminating validity/rights checking by default, and by tuning microkernel to HW
    - Only 5% degradation
  - *Exokernel* merely provides efficient multiplexing of hardware resources
    - Distributed resource management
    - Extremely fast

Case Studies

- Architecture of Unix
- The Kernel of Linux
- The Kernel of Solaris
- Architecture of Windows

Architecture of Unix

- Original Unix kernel was monolithic
- Kernel modules were added later

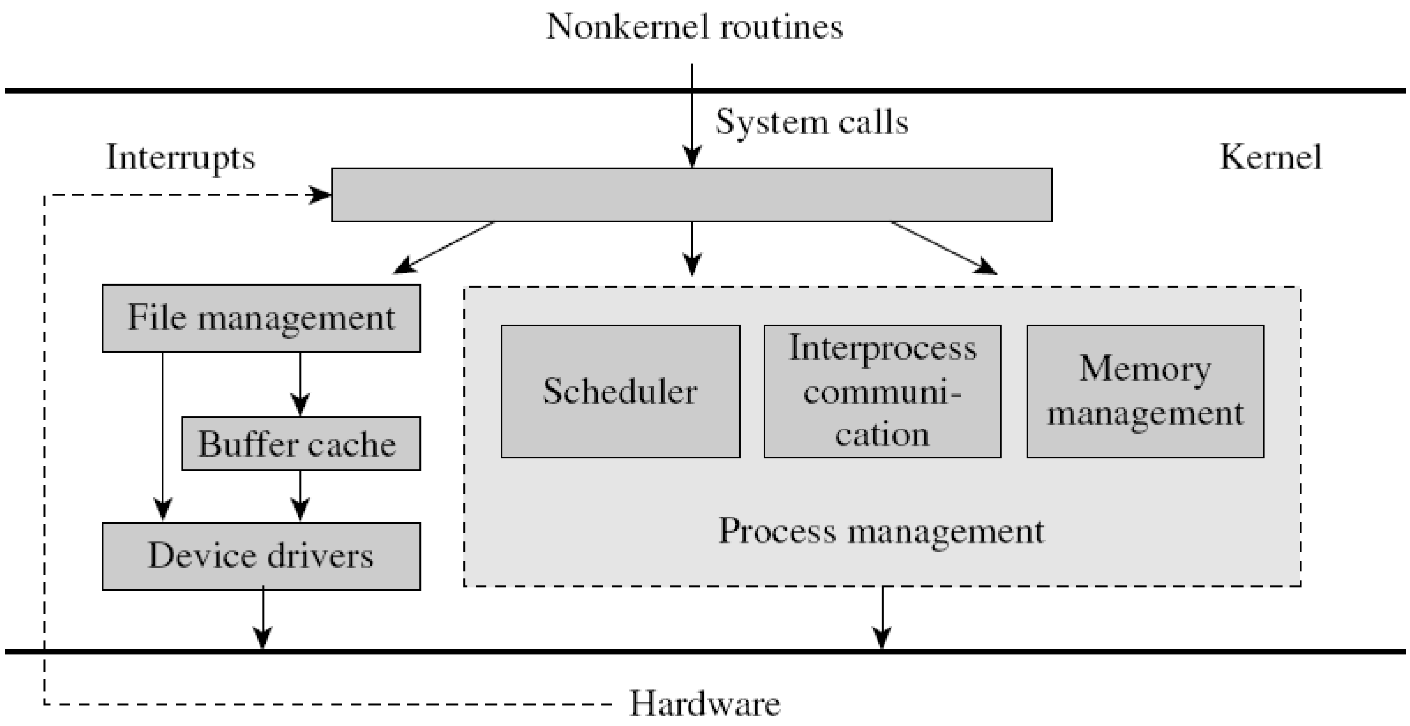

Figure 4.8 Kernel of the Unix operating system.

The Kernel of Linux

- Provides functionalities of Unix System V and BSD
- Compliant with the POSIX standard
- Monolithic kernel
- Individually loadable modules
  - A few kernel modules are loaded on boot
- Improvements in Linux 2.6 kernel:
  - Kernel is preemptible
    - More responsive to users and application programs
  - Supports architectures that do not possess a MMU
  - Better scalability through improved model of threads

The Kernel of Solaris

- SunOS was based on BSD Unix
- Solaris is based on Unix SVR4
- Since 1980s, Sun has focused on networking and distributed computing
  - Features have become standards
    - RPC
    - NFS
- Later, Sun focused on multiprocessor systems too
  - Multithreading the kernel, making it preemptible
  - Fast synchronization techniques in the kernel

The Kernel of Solaris (continued)

- Solaris 7 employs the kernel-design methodology of dynamically loadable kernel modules
  - Supports seven types of loadable modules:
    - Scheduler classes
    - File systems
    - Loadable system calls
    - Loaders for different formats of executable files
    - Streams modules
    - Bus controllers and device drivers
    - Miscellaneous modules
  - Provides easy extensibility

Architecture of Windows

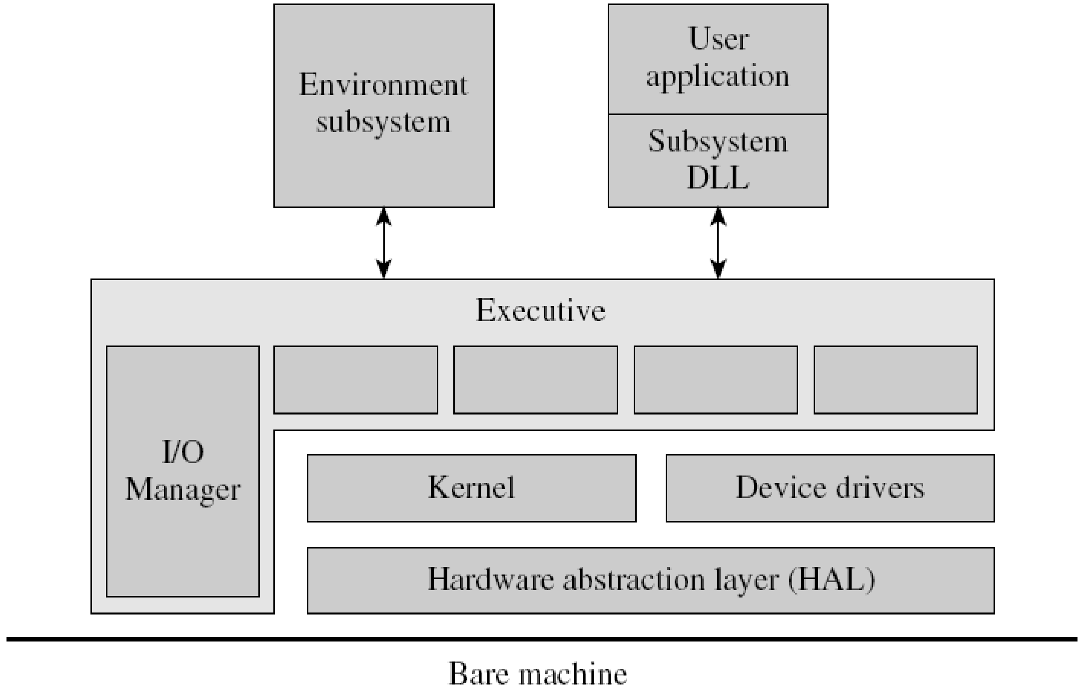

Figure 4.9 Architecture of Windows.

- HAL interfaces with the bare machine
- Environment subsystems support execution of programs written for MS DOS, Win 32 and OS/2

Summary

- *Portability:* ease with which the OS can be implemented on a computer having a different architecture
- *Extensibility:* ease with which its functionalities can be modified or enhanced to adapt it to a new computing environment
- An OS functionality typically contains a *policy*, and a few *mechanisms*
- Early OSs had a *monolithic* structure

Summary (continued)

- *Layered design* used the principle of abstraction to control complexity of designing the OS
- The *virtual machine operating system* (VM OS) supported operation of several OSs on a computer simultaneously
  - Create a *virtual machine* for each user
- In a *kernel-based* design, *kernel* is the core of the OS, which invokes the *nonkernel routines* to implement operations on processes and resources
- A *microkernel* is the essential core of OS code
  - Policy modules implemented as *server* processes

Queries ….?