**Syllabus**

**UNIT I**

**Introduction:** What Operating Systems Do, Operating-System Structure, Operating-System Operations, Process Management, Memory Management, Storage Management, Protection and Security, Kernel Data Structures.

**System Structures:** Operating-System Services, User and Operating-System Interface, System Calls, Types of System Calls, Operating-System Structure.

**Process Concept:** Process Concept, Process Scheduling, Operations on Processes, Inter process Communication.

**UNIT II**

**Multithreaded Programming:** Overview of Multithreading, Multicore Programming, Multithreading Models, Implicit Threading, Threading Issues.

**Process Scheduling:** Basic Concepts, Scheduling Criteria, Scheduling Algorithms, Thread Scheduling, Multiple-Processor Scheduling, Real-Time CPU Scheduling.

**Synchronization:** Background, The Critical-Section Problem, Peterson ‘solution, Synchronization Hardware, Mutex Locks, Semaphores, Classic Problems of Synchronization, Monitors.

**UNIT III**

**Dead Locks:** System Model, Deadlock Characterization, Methods for Handling Deadlocks, Deadlock Prevention, Deadlock Avoidance, Deadlock Detection, Recovery from Deadlock

**Memory-Management Strategies:** Background, Swapping, Contiguous Memory Allocation, Segmentation, Paging, Structure of Page Table.

**Virtual-Memory Management:** Background, Demand Paging, Page Replacement, allocation of frames, Thrashing.

**UNIT IV**

**Files System:** File Concept, Access Methods, Directory and Disk Structure, File-System Mounting, File sharing, Protection.

**Implementing File-Systems:** File-System Structure, File-System Implementation, Directory Implementation, Allocation Methods, and Free-Space Management.

**Mass-Storage Structure:** Overview of Mass-Storage Structure, Disk Structure, Disk Scheduling.

**Text Book:**

1. Operating System Concepts-Abraham Silberchatz, Peter B, Galvin, Greg Gange 9th Edition, John Wiley.

**Reference Books:**

1. Operating Systems, Internal and Design Principles, Stallings, 8th Edition-2015, Pearson education/PHI.

2. Operating system A Design Approach-Crowley, TMH.

3. Modern Operating Systems, Andrew S Tenenbaum 4th Edition Pearson/PHI.

4. An Introduction to Operating Systems, Concepts and Practice, 4th Edition, PHI, 2013- Pramod Chandra P. Bhatt.

5. Operating Systems- A concept based approach –DM Dhamdhere -3rd Edition TMH.

**Course Objectives:** At the end of the course, the student will understand

1. Operating system structure, functions and IPC mechanism.

2. Concepts of multithreading, process scheduling and process synchronization.

3. Dead lock handling mechanisms and memory management techniques.

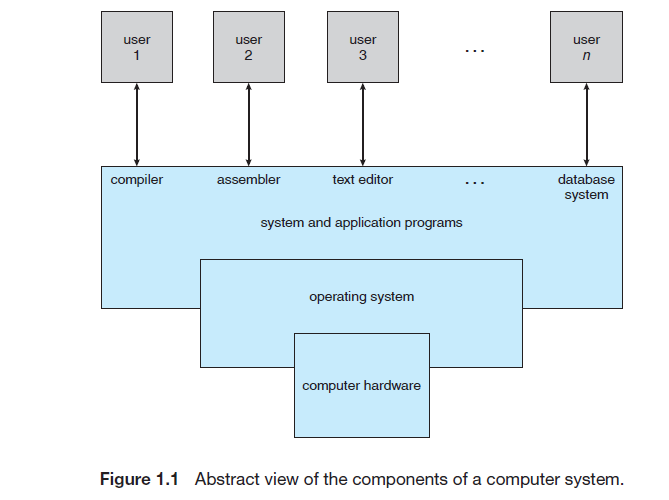
4. Concepts of file management and secondary storage management.

**Operating System:**

An operating systemis a program that manages the computer hardware. It also provides a basis for application programs and acts as an intermediary between the computer user and the computer hardware.

Operating systems are designed primarily to optimize utilization of hardware.

**What Operating Systems Do:**



A computer system can be divided roughly into four components: the hardware, the operating system, the application programs and the users. It is shown in (Figure 1.1).

**The hardware:** the central processing unit (CPU), the memory, and the input/output (I/O) devices—provides the basic computing resources for the system.

**The application programs**—such as word processors, spreadsheets, compilers, and Web browsers—define the ways in which these resources are used to solve users’ computing problems. The operating system controls the hardware and coordinates its use among the various application programs forthe various users.

**The operating system** controls the hardware and coordinates its use among the various application programs for the various users.

Computer system as consisting of hardware, software, and data. The operating system provides the means for proper use of these resources in the operation of the computer system. An operating system is similar to a government.

To understand more fully the operating system’s role, in User’s view and System’s view

**User View:**

In this case, the operating system is designed mostly for ease of use, with some attention paid to performance and none paid to resource utilization—how various hardware and software resources are shared. Performance is, of course, important to the user; but such systems are optimized for the single-user experience rather than the requirements of multiple users.

Other users are accessing the same computer through other terminals. These users share resources and may exchange information. The operating system in such cases is designed to maximize resource utilization—to assure that all available CPU time, memory, and I/O are used efficiently and that no individual user takes more than her fair share.

**System View:**

In this case operating system acts as a resource allocator. The operating system must decide how to allocate resources to specific programs and users so that it can operate the computer system efficiently and fairly.

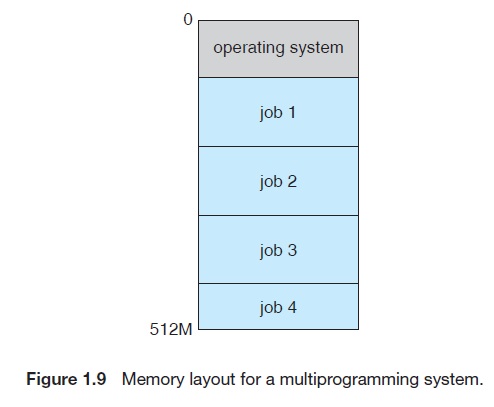
An operating system is a control program. A control program manages the execution of user programsto prevent errors and improper use of the computer.

OS is the one program running at all times on the computer” is the kernel. Along with the kernel, there are two other types of programs: systems programs, which are associated with the operating system but are not part of the kernel, and application programs, which include all programs notassociated with the operation of the system.

**Operating System Structure:**

An operating system provides the environment within which programs areexecuted. Internally, operating systems vary greatly in their makeup, sincethey are organized along many different lines. There are, however, manycommonalities, which we consider in this section.

* One of the most important aspects of operating systems is the abilityto multiprogram
* In multiprogramming multiple programs are executed. So the CPUor the I/O devices busy at all times
* Multiprogramming increases CPUutilization by organizingjobs (code and data) so that the CPU always has one to execute.
* The operating system keeps several jobs in memorySimultaneously. It is shown in fig.



* In general, main memory is too small toaccommodate all jobs, the jobs are kept initially on the disk in the job pool.This pool consists of all processes residing on disk awaiting allocation of mainmemory.
* The set of jobs in memory can be a subset of the jobs kept in the job pool.The operating system picks and begins to execute one of the jobs in memory.Eventually, the job may have to wait for some task, such as an I/O operation,to complete. In a non-multiprogrammed system, the CPU would sit idle. Ina multiprogrammed system, the operating system simply switches to, andexecutes, another job. When that job needs to wait, the CPU is switched toanother job, and so on. Eventually, the first job finishes waiting and gets theCPU back. As long as at least one job needs to execute, the CPU is never idle.

**Time Sharing (Multitasking):**

* It is a logical extension ofmultiprogramming. In time-sharing systems, the CPU executes multiple jobsby switching among them, but the switches occur so frequently that the userscan interact with each program while it is running.
* Time sharing requires an interactive (or hands-on) computer system,which provides direct communication between the user and the system. Theuser gives instructions to the operating system or to a program directly, using ainput device such as a keyboard or a mouse, andwaits for immediate results onan output device. Accordingly, the response time should be short—typicallyless than one second.
* A time-shared operating system allows many users to share the computersimultaneously. Since each action or command in a time-shared system tendsto be short, only a little CPU time is needed for each user. As the system switchesrapidly from one user to the next, each user is given the impression that the entire computer system is dedicated to his use, even though it is being shared among many users.
* Aprogram loaded intomemory and executing is called a process

**Difference between multiprogramming and Time sharing:**

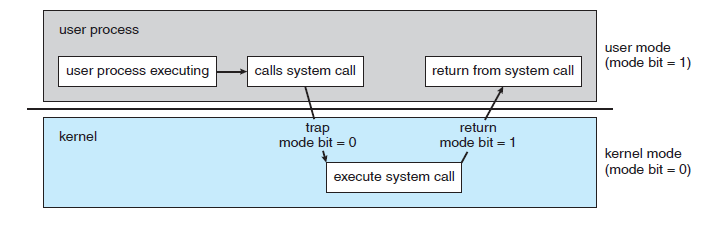
|  |  |
| --- | --- |
| multiprogramming | Time sharing |
| Multiprogramming operating system allows to execute multiple processes by monitoring their process states and switching in between processes. | Time Sharing is the logical extension of multiprogramming, in this time sharing Operating system many users/processes are allocated with computer resources in respective time slots |
| Processor and memory underutilization problem is resolved and multiple programs runs on CPU that’s why it is called multiprogramming. | Processors time is shared with multiple users that’s why it is called as time sharing operating system. |
| Multi-programming OS has no fixed time slice. | Time sharing OS has fixed time slice. |
| In multiprogramming the system does not take same time to work on different processes. | In multitasking the system works for the same or less time on each processes. |
| System model of multiprogramming system is multiple programs. | System model of time sharing system is multiple programs and multiple users. |
| maximizes response time | maximizes response time |

**Difference between program and process**

|  |  |
| --- | --- |
| Process | Program |
| Process is an instance of an executing program. | Program contains a set of instructions designed to complete a specific task. |
| Process is a active entity as it is created during execution and loaded into the main memory. | Program is a passive entity as it resides in the secondary memory |
| Process exists for a limited span of time as it gets terminated after the completion of task. | Program exists at a single place and continues to exist until it is deleted. |
| Process is a dynamic entity. | Program is a static entity. |
| Process has a high resource requirement, it needs resources like CPU, memory address, I/O during its lifetime. | Program does not have any resource requirement, it only requires memory space for storing the instructions. |
| Process has its own control block called Process Control Block. | Program does not have any control block |

**Operating-System Operations:**

* An error in one program can adversely affect many processes, it might modify data of another program, or also can affect the operating system. For example, if a process stuck in infinite loop then this infinite loop could affect correct operation of other processes. So to ensure the proper execution of the operating system, there are two modes of operation:
* user mode and kernel mode (also called supervisor mode, system mode, or privilegedmode).A bit, called the mode bit, is added to the hardware of the computer toindicate the current mode: kernel (0) or user (1).
* When the computer system isexecuting on behalf of a user application, the system is in user mode.However,when a user application requests a service from the operating system (via asystem call), it must transition from user to kernel mode to fulfill the request.This is shown in Figure



* At system boot time, the hardware starts in kernel mode. The operatingsystem is then loaded and starts user applications in user mode. Whenever atrap or interrupt occurs, the hardware switches from user mode to kernel mode(that is, changes the state of the mode bit to 0). Thus, whenever the operatingsystem gains control of the computer, it is in kernel mode. The system alwaysswitches to user mode (by setting the mode bit to 1) before passing control toa user program.

**Timer:** We cannot allow a user program to get stuck in an infinite loop or to fail to call system services and never return control to the operating system. To accomplish this goal, we can use a timer. A timer can be set to interrupt the computer after a specified period. The period may be fixed (for example, 1/60 second) or variable (for example, from 1 millisecond to 1 second).

**Process Management:** The operating system is responsible for the following activities in connection with process management:

• Scheduling processes and threads on the CPUs

• Creating and deleting both user and system processes

• Suspending and resuming processes

• Providing mechanisms for process synchronization

• Providing mechanisms for process communication

**Memory Management:**

* The operating system is responsible for the following activities in connection with memory management:
* Keeping track of which parts of memory are currently being used and by whom
* Deciding which processes (or parts thereof) and data to move into and out of memory
* Allocating and deallocating memory space as needed

**Storage Management:**

The operating system maps files onto physical media and accesses these files via the storage devices.

File System Management

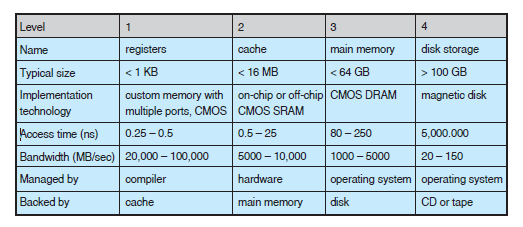
The operating system is responsible for the following activities in connection with file management:

* The operating system is responsible for the following activities in connection with file management:
* Creating and deleting files
* Creating and deleting directories to organize files
* Supporting primitives for manipulating files and directories
* Mapping files onto secondary storage
* Backing up files on stable (nonvolatile) storage media

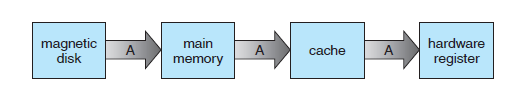
Mass-Storage Management:

* The operating system is responsible for the following activities in connection with disk management:
* Free-space management
* Storage allocation
* Disk scheduling

Performance of Various Levels of Storage is shown in following table



Example: Migration of Integer A from disk to Register

* Multitasking environments must be careful to use most recent value, no matter where it is stored in the storage hierarchy.
*  Multiprocessor environment must provide cache coherency in hardware such that all CPUs have the most recent value in their cache.
* Distributed environment situation even more complex.

**Protection and Security**

* **Protection** – any mechanism for **controlling access** (of processes or users) to resources defined by the OS
  + Files access, memory, Device-control registers, timers
* **Security** – defense of the system against **internal and external attacks**
  + Examples: denial-of-service, worms, viruses, identity theft, theft of service
* Systems generally first distinguish among users, to determine who can do what
  + User identities (**user IDs**, security IDs) include name and associated number, one per user
  + User ID then **associated with all files, processes** of that user to determine access control
  + Group identifier (**group ID**) allows set of users to be defined and controls managed, then also associated with each process, file

**Kernal Data Structures:**

The kernel data structures play a majorl role because they store data about the current state of the system.The following data structures are used.

**Single Linked List:**

**It is shown in following figure**

* Single Linked List: It is a linear data structure, contains data and pointer to next node. It is shown in following figure.

1_13.pdf

* Double Linked List: It is a linear data structure, contains data,pointer to next node, pointer to previous node. It is shown in following figure.

1_14.pdf

* Circular linked List: It is similar to single linked list. But last node points to the first node. It is shown in following figure.

1_15.pdf

* Binary Search Tree: It is non linear data structure. It is binary tree and the satisfies the following properties
* The value of the key of the left sub-tree is less than the value of its parent (root) node's key.
* The value of the key of the right sub-tree is greater than or equal to the value of its parent (root) node's key.

Example:

1_16.pdf

* **Hash function** can create a **hash map**

The hash function is h(x)=x%n where n is table size. It is shown in figure

1_17.pdf

**Bitmap** – In this data structure string of *n* binary digits representing the status of *n* items.

**CHAPTER 2**

**Operating System Services:**

**User interface:** operating systems have a **user interface.** This interface can take several forms.

* **Command-line interface(CLI)**, which uses text commands and a method for entering them.
* **Batch interface**, in which commands and directives to control those commands are entered into files, and those files are executed.
* **graphical user interface (GUI)** is used. Here, the interface is a window system with a pointing device to direct I/O, choose from menus, and make selections and a keyboard to enter text.
* **Program execution:**The system must be able to load a program into
* memory and to run that program. The program must be able to end its execution, either normally or abnormally.
* **I/O operations:** A running program may require I/O, which may involve a file or an I/O device. Then operating system must provide a I/O.
* **File-system manipulation:** programs need to read and write files and directories, create and delete them, search them, list file Information, permission management to allow or deny access to files or directories based on file ownership.
* **Communications:** Processes may exchange information, on the same computer or between computers over a network Communications may be via shared memory or through message passing.
* **Error detection:** Errors may occur in the CPU and memory hardware (such as a memory error or a power failure), in I/O devices (such as a parity error on tape, a connection failure on a network, or lack of paper in the printer), and in the user program. For each type of error, the operating system should take the appropriate action.
* **Resource allocation**: When there are multiple users or multiple jobs running at the same time, resources must be allocated to each of them. Many different types of resources are managed by the operating system.
* **Accounting:** To keep track of which users use how much and what kinds of computer resources.
* **Protection and security**: The owners of information stored in a multiuser or networked computer system may want to control use of that information.

**The OS services shown in following figure**.



**User Operating-System Interface:**

**Command Line Interpreter:** that allows users to directly enter commands to be performed by the operating system. The main function of the command interpreter is to get and execute the next user-specified command. Many of the commands given at this level manipulate files: *create*, *delete*, *list*, *print*, *copy*, *execute*, and so on. The MS-DOS and UNIX shells operate in this way. These commands can be implemented in two general ways. The command interpreter itself contains the code to execute the command. For example, a command to delete a file may cause the command interpreter to jump to a section of its code that sets up the parameters and makes the appropriate system call.

Other way is implements most commands through system programs.

rm file.txt

**Graphical User Interfaces:**

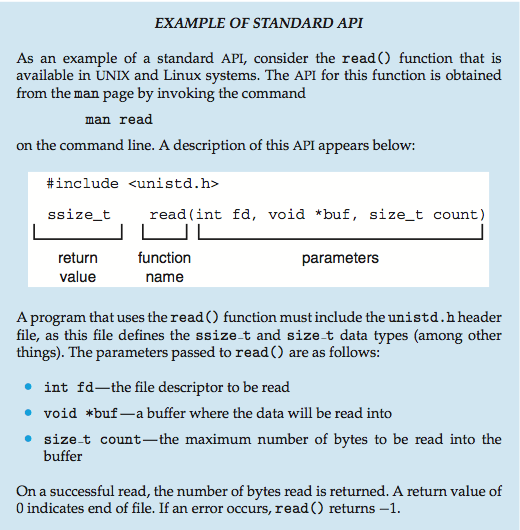
Here, rather than entering commands directly via a command-line interface, users employ a mouse-based window and-menu system characterized by a **desktop** metaphor. The user moves the mouse to position its pointer on images, or **icons**, on the screen (the desktop) that represent programs, files, directories, and system functions. Depending on the mouse pointer’s location, clicking a button on the mouse can invoke a program, select a file or directory—known as a **folder**—or pull down a menu that contains commands.

**System Calls:**

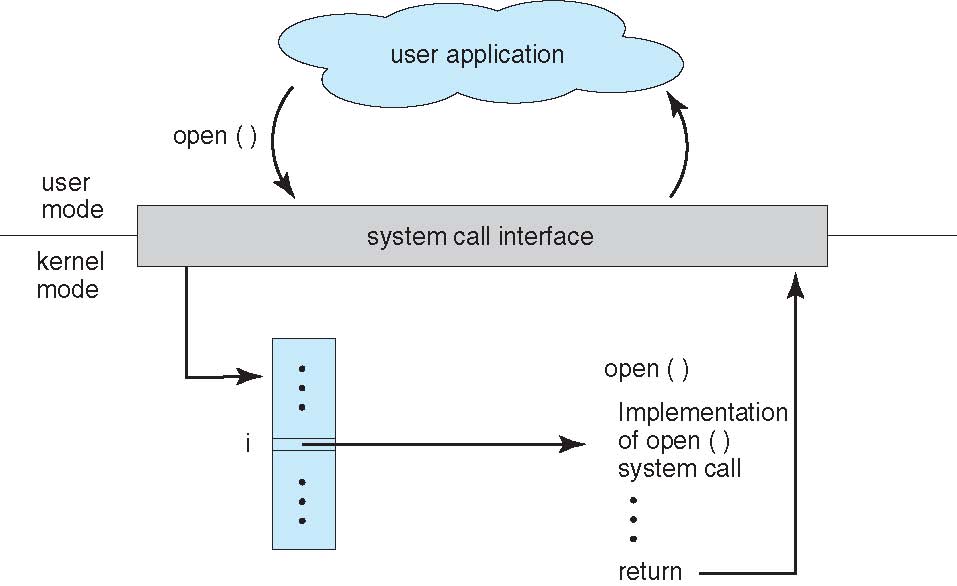
System calls provide an interface to the services made available by an operating system. These calls are generally available as routines written in C and C++, although certain low-level tasks (for example, tasks where hardware must be accessed directly) may need to be written using assembly-language instructions.

System call sequence to copy the contents of one file to another file





API, the System Call interface and OS Relationship is shown in figure which illustrates how the operating system handles a user application invoking the open() system call.

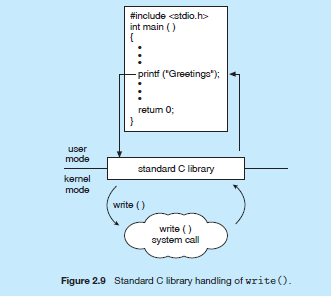


Types of System calls:

* Process control
  + create process, terminate process
  + end, abort
  + load, execute
  + get process attributes, set process attributes
  + wait for time
  + wait event, signal event
  + allocate and free memory
  + Dump memory if error
  + **Debugger** for determining **bugs, single step** execution
  + **Locks** for managing access to shared data between processes
* File management
  + create file, delete file
  + open, close file
  + read, write, reposition
  + get and set file attributes
* Information maintenance
  + get time or date, set time or date
  + get system data, set system data
  + get and set process, file, or device attributes
* Communications
  + create, delete communication connection
  + send, receive messages if **message passing model** to **host name** or **process name**
    - From **client** to **server**
  + **Shared-memory model** create and gain access to memory regions
  + transfer status information
  + attach and detach remote devices
* Protection
  + Control access to resources
  + Get and set permissions
  + Allow and deny user access
* **EXAMPLES OF WINDOWS AND UNIX SYSTEM CALLS**

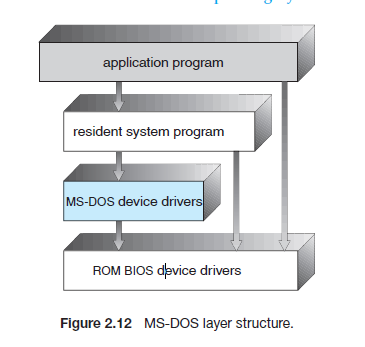
|  |  |  |
| --- | --- | --- |
| **System call** | **Windows** | **Unix** |
| Process Control | CreateProcess()  ExitProcess()  WaitForSingleObject() | fork()  exit()  wait() |
| File Manipulation | CreateFile()  ReadFile()  WriteFile()  CloseHandle() | open()  read()  write()  close() |
| Device Manipulation | SetConsoleMode()  ReadConsole()  WriteConsole() | ioctl()  read()  write() |
| **Information Maintanance** | GetCurrentProcessID()  SetTimer()  Sleep() | getpid()  alarm()  sleep() |
| **Communication** | CreatePipe()  CreateFileMapping()  MapViewOfFile() | pipe()  shmget()  mmap() |
| **Protection** | SetFileSecurity()  InitializeSecurityDescriptor()  SetSecurityDescriptorGroup() | chmod()  umask()  chown() |

Example: standard C library handling of write

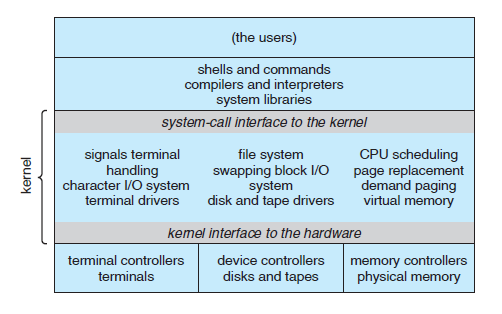


**Operating System Structure:**

* Simple Structure: It is a small, simple, and limited systems.
* MS-DOS is an example
* It was originally designed and implemented by a few people who had no idea that it would become so popular.
* no dual mode and no hardware protection



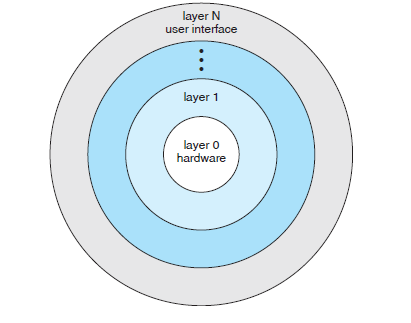
* It is further separated into a series of interfaces and device drivers. Ex unix system structure. Traditional UNIX structure shown in fig.



* **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 *N*) is the user interface. It is shown in following figure.

Advantage

* + simplicity of construction and debugging.



Disadvantage:

* The major difficulty with the layered approach involves appropriately defining the various layers. Because a layer can use only lower-level layers, careful planning is necessary.

Microkernels:

* In unix the kernel became large and difficult to manage.
* In the mid-1980s, researchers at Carnegie Mellon University developed an operating system called **Mach** that modularized the kernel using the **microkernel** approach.
* removing all nonessential components from the kernel and implementing them as system and user-level programs.
* Communication takes place between user modules using message passing

Benefits:

* Easier to extend a microkernel
* Easier to port the operating system to new architectures
* More reliable (less code is running in kernel mode)
* More secure

Detriments:

Performance decreased

Modules:

* Most modern operating systems implement kernel modules
* Uses object-oriented approach
* Each core component is separate
* Each talks to the others over known interfaces
* Each is loadable as needed within the kernel
* Overall, similar to layers but with more flexible

Example: Solaris Modular approach which is shown in following figure

It has seven types of loadable modules.

* Scheduling classes
* File systems
* Loadable system calls
* Executable formats
* Streams modules
* Miscellaneous
* Device and bus drivers

