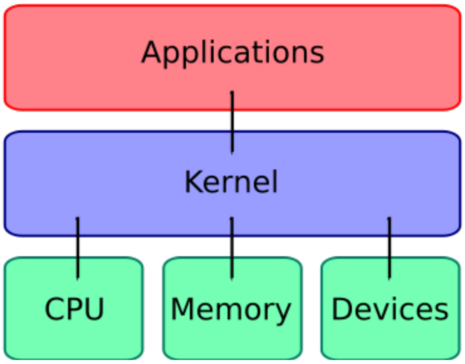
## What is an Operating System?

- is a collection of system programs that together control the operation of a computer system.
- is responsible for creating a link between the material resources, the user and the applications.
- is the infrastructure software component of a computer system responsible for the management and coordination of activities and the sharing of the resources of the computer.
- offers a simplified man-machine interface (MMI) to overcome the complexity of actual machine.
- The core of the operating system is the kernel, a control program that functions in privileged state (an execution context that allows all hardware instructions to be executed), reacting to interrupts from external devices and to service requests and traps from processes. Generally, the kernel is a permanent resident of the computer. It creates and terminates processes and responds to their request for service.



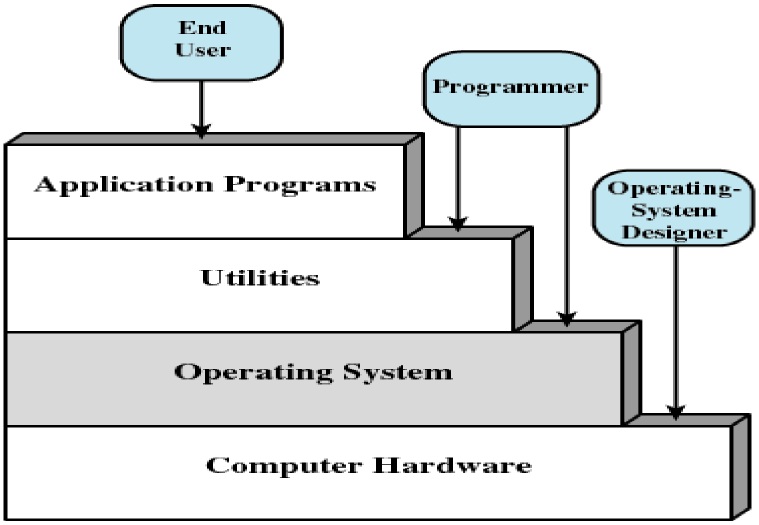


**Figure 2.1 Layers and Views of a Computer System**

## Roles Of OS

- Implementing the user interface
- Sharing hardware among users
- Allowing users to share data among themselves
- Preventing users from interfering with one another
- Scheduling resources among users
- Facilitating input/output
- Recovering from errors
- Accounting for resource usage
- Facilitating parallel operations
- Organizing data for secure and rapid access
- Handling network communications

## Two Functions of OS

1. OS as an extended machine
    - OS creates a high level of abstraction for the programmer
        - Example—(Floppy Disk I/O Operation)
            - Disk contains a list of named files
            - Each file can be opened for read write
            - After read write is complete, close that file
            - No any detail to deal
    - The abstraction hides the truth about the hardware from the programmer and presents a simple view of the named files that can be read and written.
    - OS prevents the user with the equivalent of an extended machine or virtual machine that is easier to program than the underlying hardware.
2. OS as a Resource Manager
    - What happens if three programs try to print their output on the same printer at the same time
    - What happens if two network users try to update a shared document at the same time?
    - OS primary function is to keep track of who is using which resource to grant resource requests, to account for usage and to mediate conflicting requests from different programs and users
    - Virtualizes resources so multiple users can share it
    - Protect application from one another
    - Provide efficient and fair access to resource

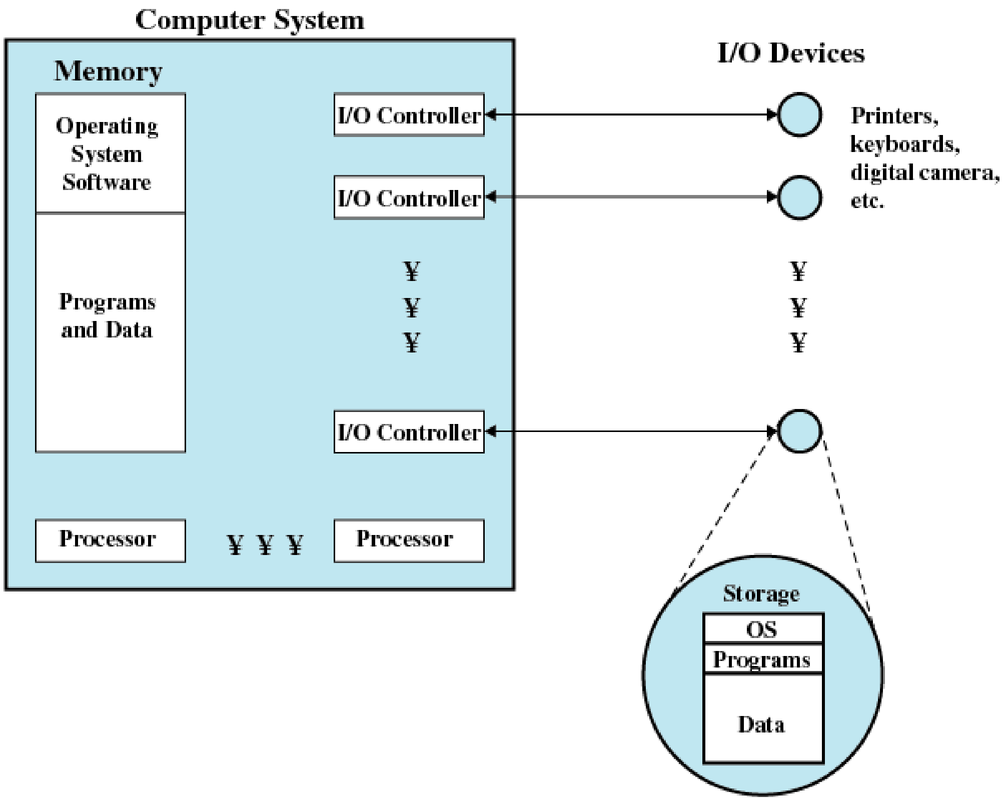


Figure 2.2 The Operating System as Resource Manager

## History of OS

Operating systems have evolved through a number of distinct phases or generations which corresponds roughly to the decades.

**The 1940's - First Generations (User Driven)**

- The earliest electronic digital computers had no operating systems.
- Machines of the time were so primitive that programs were often entered one bit at time on rows of mechanical switches (plug boards).
- Programming languages were unknown (not even assembly languages).
- Operating systems were unheard of.

**The 1950's - Second Generation (Batch Processing)**

- By the early 1950's, the routine had improved somewhat with the introduction of punch cards.
- The General Motors Research Laboratories implemented the first operating systems in early 1950's for their IBM 701.
- The system of the 50's generally ran one job at a time. These were called single-stream batch processing systems because programs and data were submitted in groups or batches.

**The 1960's - Third Generation (Multiprogramming)**

- The systems of the 1960's were also batch processing systems, but they were able to take better advantage of the computer's resources by running several jobs at once.
- So operating systems designers developed the concept of **multiprogramming** in which several jobs are in main memory at once; a processor is switched from job to job as needed to keep several jobs advancing while keeping the peripheral devices in use.
- For example, on the system with no multiprogramming, when the current job paused to wait for other I/O operation to complete, the CPU simply sat idle until the I/O finished. The solution for this problem that evolved was to partition memory into several pieces, with a different job in each partition. While one job was waiting for I/O to complete, another job could be using the CPU.
- Another major feature in third-generation operating system was the technique called **spooling** (simultaneous peripheral operations on line). In spooling, a high-speed device like a disk interposed between a running program and a low-speed device involved with the program in input/output. Instead of writing directly to a printer, for example, outputs are written to the disk. Programs can run to completion faster, and other programs can be initiated sooner when the printer becomes available, the outputs may be printed.
- Another feature present in this generation was **time-sharing** technique, a variant of multiprogramming technique, in which each user has an on-line (i.e., directly connected) terminal. Because the user is present and interacting with the computer, the computer system must respond quickly to user requests, otherwise user productivity could suffer. Timesharing systems were developed to multiprogram large number of simultaneous interactive users.

**Fourth Generation (Client Server/Distributed Systems)**

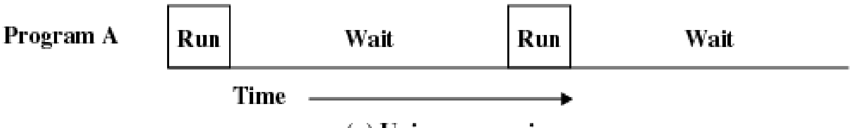
- With the development of LSI (Large Scale Integration) circuits, chips, operating system entered in the system entered in the personal computer and the workstation age.
- Microprocessor technology evolved to the point that it becomes possible to build desktop computers as powerful as the mainframes of the 1970s.

- Two operating systems have dominated the personal computer scene: MS-DOS, written by Microsoft, Inc. for the IBM PC and other machines using the Intel 8088 CPU and its successors, and UNIX, which is dominant on the large personal computers using the Motorola 6899 CPU family.

## Operating System Concepts

### Uniprogramming

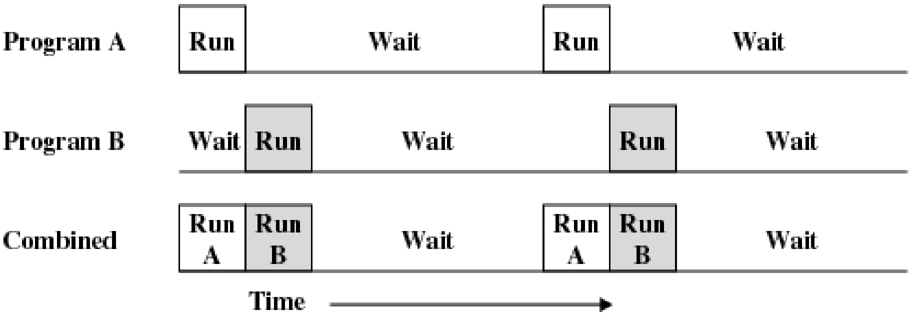
- Processor must wait for I/O instruction to complete before preceding



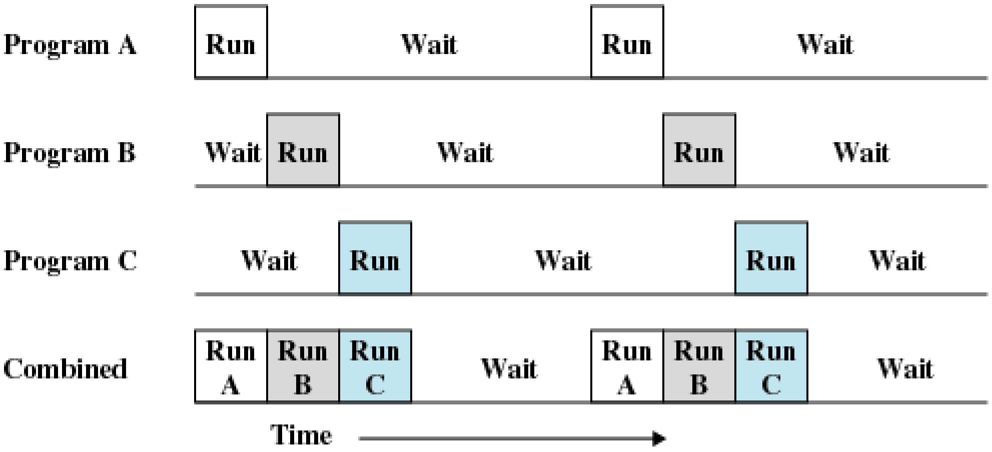
(a) Uniprogramming

### Multiprogramming

- When one job needs to wait for I/O, the processor can switch to the other job



(b) Multiprogramming with two programs



(c) Multiprogramming with three programs

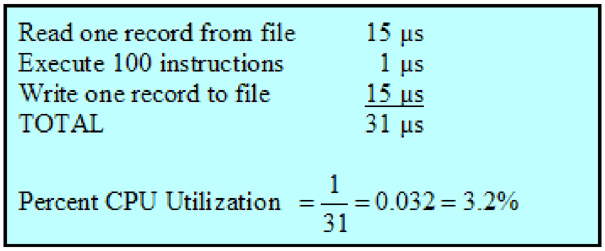
### Real Time Operating System:

- is a multitasking operating system intended for real-time applications where time is a key parameter.
- Often used as a control device in a dedicated application such as industrial robots, spacecraft, medical imaging systems, industrial control and scientific research equipment.

- A RTOS facilitates the creation of a real-time system, but does not guarantee the final result will be real-time; this requires correct development of the software.
- The completion of task before deadline is useful but extension of time after deadline crashes the entire system.
- May be either
    - Hard Real Time:
        - Well defined fixed time constraints, processing must be done within the defined constraints, or the system will fail. (e.g. use of brake in car)
    - Soft Real Time System:
        - Less stringent timing constraints and does not support the dead line scheduling
        - The value of job degrades as the dead line passes (e.g. late attendance in exam hall)

**Time Sharing Systems**

- Time-sharing is sharing a computing resource among many users by multitasking.
- By allowing a large number of users to interact simultaneously on a single computer, time-sharing dramatically lowered the cost of providing computing, while at the same time making the computing experience much more interactive.
- The CPU executes multiple jobs by switching among them, but the switches occur so frequently that the user can interact with each program while its running.
- Requires complex job scheduling and memory management

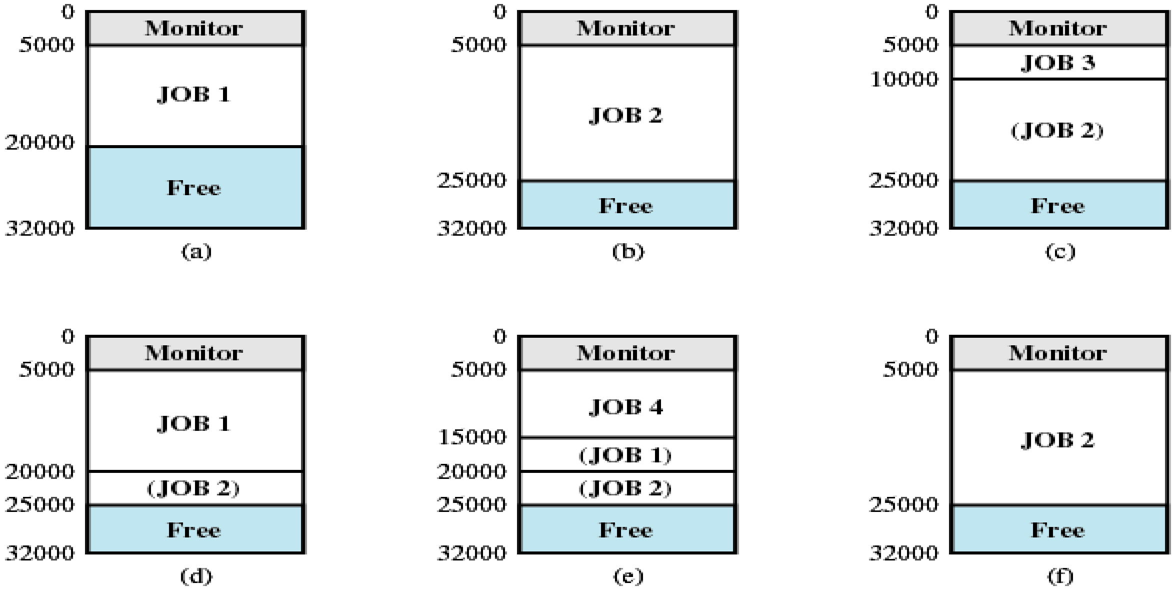


| | |
|---|---|
| Read one record from file | 15 µs |
| Execute 100 instructions | 1 µs |
| Write one record to file | 15 µs |
| TOTAL | 31 µs |

$$\text{Percent CPU Utilization} = \frac{1}{31} = 0.032 = 3.2\%$$

Figure 2.4 System Utilization Example

- **First time-sha patible Time-Sharing System - CTSS)**



A:

1(

In des
tc be
w un.
H ory,
re An
a( d).

Figure 2.7 CTSS Operation

When JOB4 is loaded, part of JOB1 and the portion of JOB2 remaining in memory are retained (e). At this point, if either JOB1 or JOB2 is activated, only a partial load will be required. In this example, it is JOB2 that runs next. This requires that JOB4 and the remaining resident portion of JOB1 be written our and that the missing portion of JOB2 be read in.

### Personal Computer Operating System

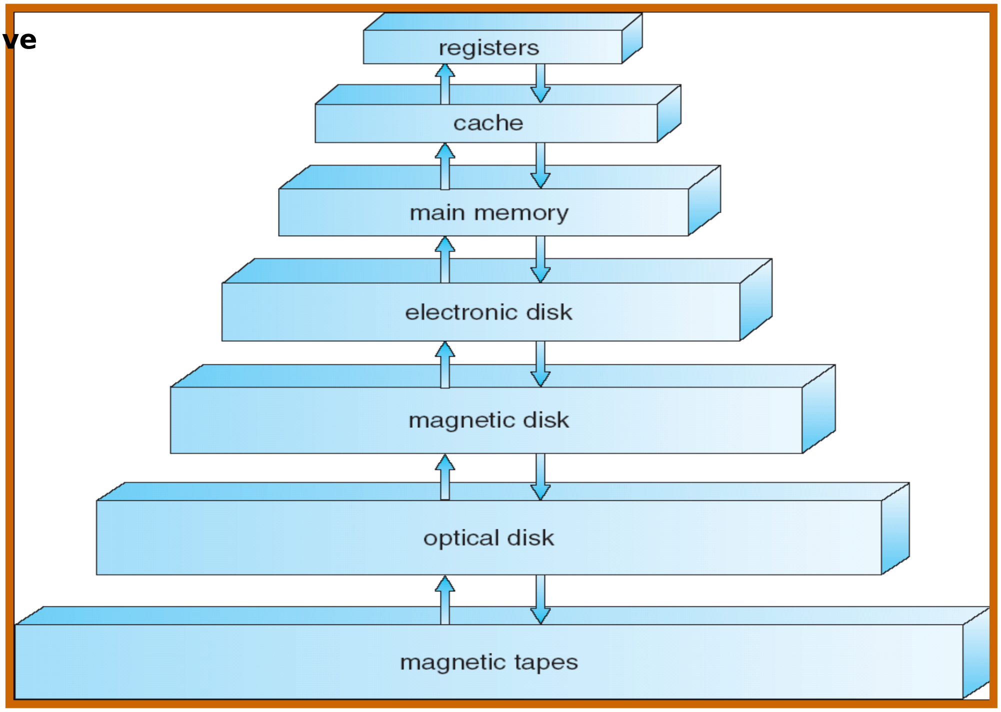
- Dedicated machine per user where CPU utilization is not a prime concern
- Hence, some of the design decisions made for advanced system may not be appropriate for these smaller systems. But other design decisions, such as those for security, are appropriate because PCs can now be connected to other computer and users through the network and the web
- There are different types of OSs for PC such as Windows, MacOS, Linux and Unix.
- OS functionality changes with hardware and users.

### Mainframe Operating System

- These computers distinguish themselves from PCs in terms of their I/O capacity.
- A mainframe with 1000 disks and thousands of gigabyte of data is not unusual.
- They may be used as high-end web servers, servers for large-scale e-commerce sites and servers for business to business transactions.
- Are heavily oriented towards processing many jobs at once, most of which need prodigious amounts of I/O.
- They typically offer three kinds of services: batch, transaction processing and timesharing.

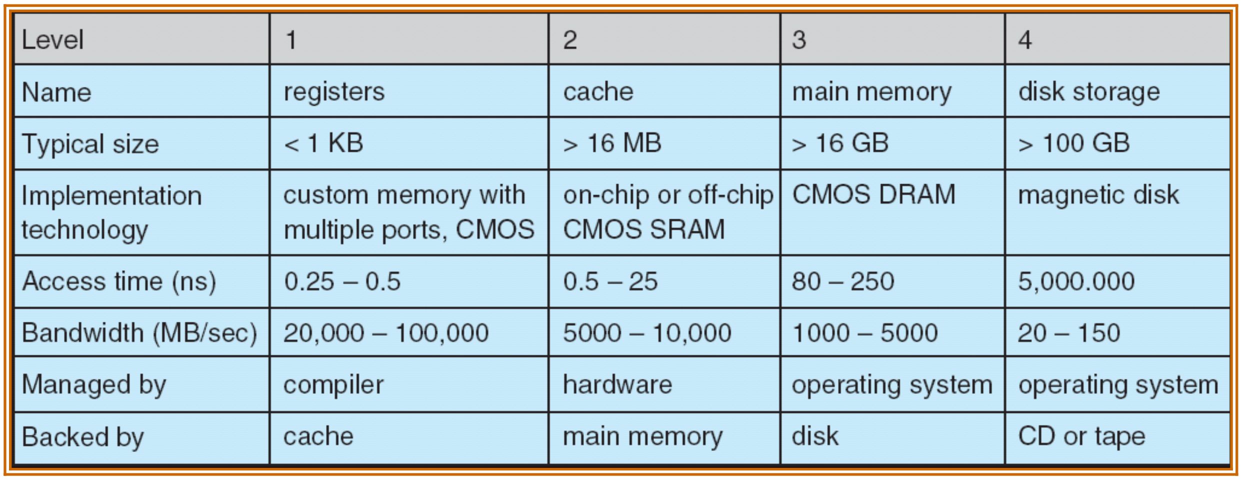
- **Memory Hierarchy**

**Fast and Expensive**



**Slow And Cheap**

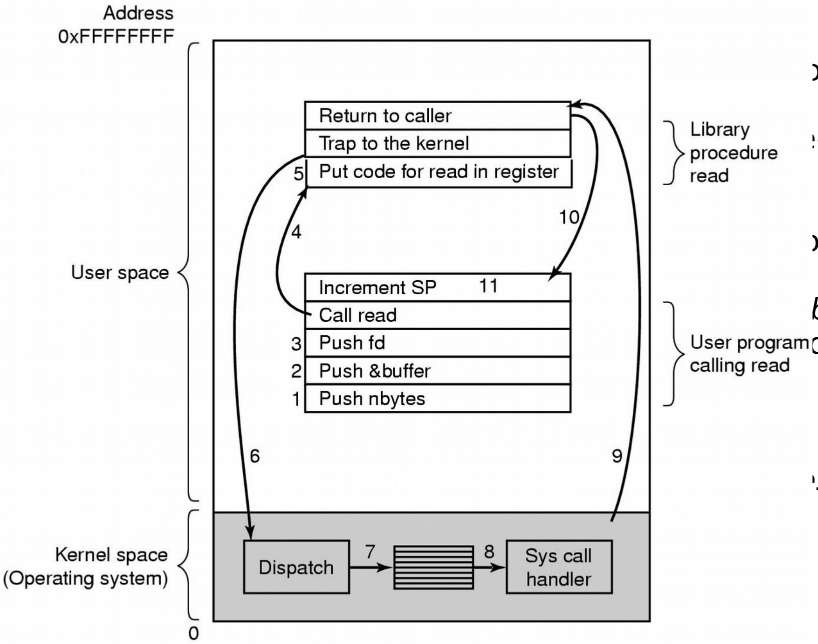
**Performance:**



| Level | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| Name | registers | cache | main memory | disk storage |
| Typical size | < 1 KB | > 16 MB | > 16 GB | > 100 GB |
| Implementation technology | custom memory with multiple ports, CMOS | on-chip or off-chip CMOS SRAM | CMOS DRAM | magnetic disk |
| Access time (ns) | 0.25 – 0.5 | 0.5 – 25 | 80 – 250 | 5,000.000 |
| Bandwidth (MB/sec) | 20,000 – 100,000 | 5000 – 10,000 | 1000 – 5000 | 20 – 150 |
| Managed by | compiler | hardware | operating system | operating system |
| Backed by | cache | main memory | disk | CD or tape |

**Introduction to**

- A system ca request service from the ker
- The interfac d by the set of systems call
- Generally av
- Languages ( gramming allow system calls
- UNIX has a s *buffer,nbytes)*
- One to spec d one to tell how many bytes

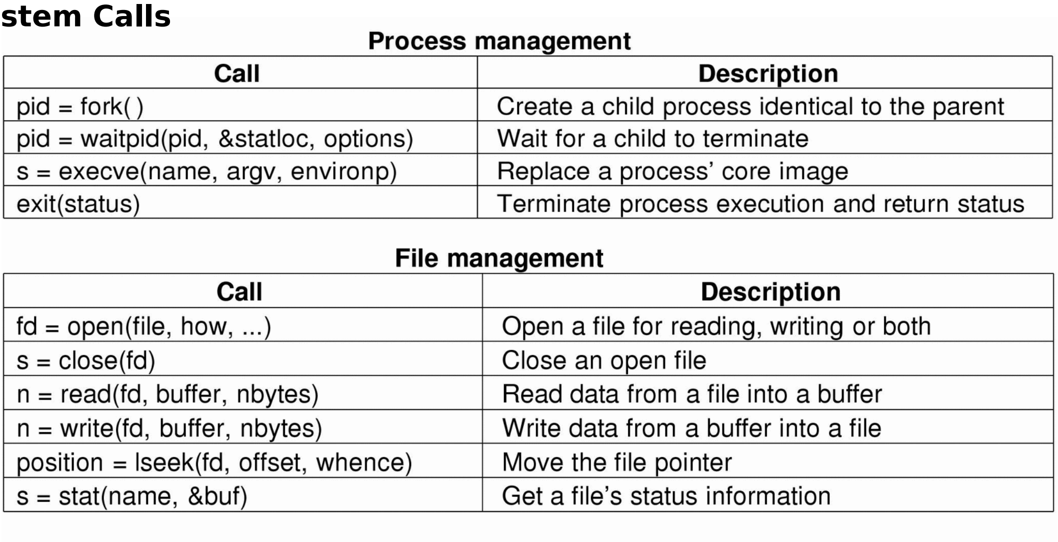


**Steps in Making**

- There are 11 *s)*

- Push parameter into the stack (1-3)
- Calls library procedure (4)
- Pass parameters in registers (5)
- Switch from user mode to kernel mode and start to execute (6)
- Examine the system call number and then dispatch to the correct system call handler via a table of pointer (7)
- Run System call Handlers (8)
- Once the system call handler completed its work, control return to the library procedure (9)
- This procedure then return to the user program in the usual way (10)
- Increment SP before call to finish the job. (11)

## Types of System Calls

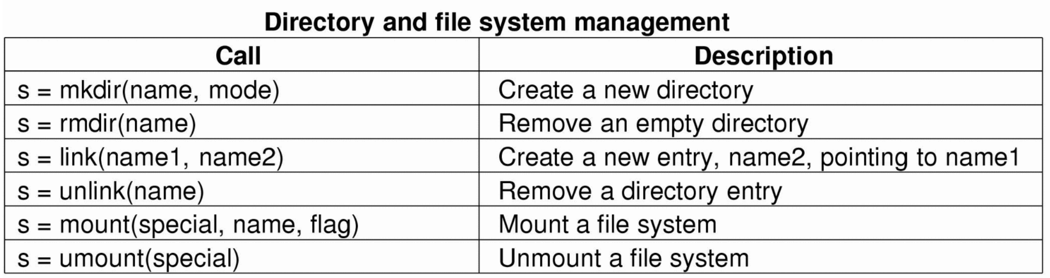


**Process management**

| Call | Description |
|---|---|
| pid = fork( ) | Create a child process identical to the parent |
| pid = waitpid(pid, &statloc, options) | Wait for a child to terminate |
| s = execve(name, argv, environp) | Replace a process' core image |
| exit(status) | Terminate process execution and return status |

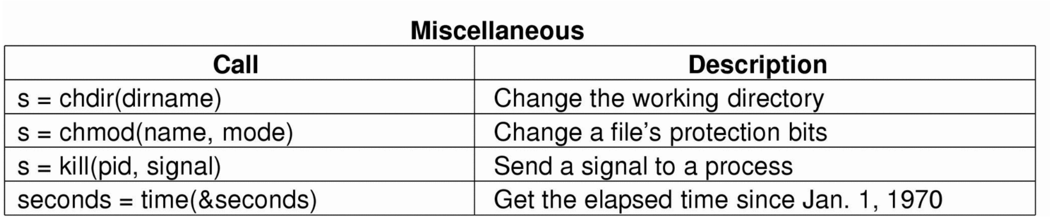
**File management**

| Call | Description |
|---|---|
| fd = open(file, how, ...) | Open a file for reading, writing or both |
| s = close(fd) | Close an open file |
| n = read(fd, buffer, nbytes) | Read data from a file into a buffer |
| n = write(fd, buffer, nbytes) | Write data from a buffer into a file |
| position = lseek(fd, offset, whence) | Move the file pointer |
| s = stat(name, &buf) | Get a file's status information |



Directory and file system management

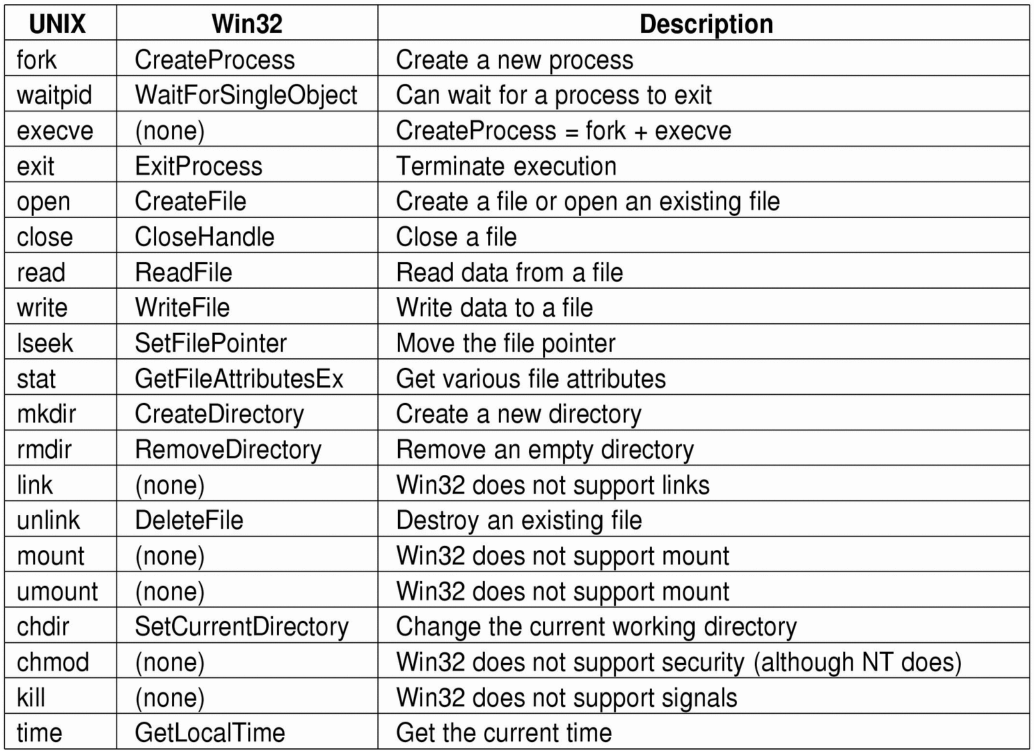
| Call | Description |
|---|---|
| s = mkdir(name, mode) | Create a new directory |
| s = rmdir(name) | Remove an empty directory |
| s = link(name1, name2) | Create a new entry, name2, pointing to name1 |
| s = unlink(name) | Remove a directory entry |
| s = mount(special, name, flag) | Mount a file system |
| s = umount(special) | Unmount a file system |



Miscellaneous

| Call | Description |
|---|---|
| s = chdir(dirname) | Change the working directory |
| s = chmod(name, mode) | Change a file's protection bits |
| s = kill(pid, signal) | Send a signal to a process |
| seconds = time(&seconds) | Get the elapsed time since Jan. 1, 1970 |

### Some Win32 API Calls



| UNIX | Win32 | Description |
|---|---|---|
| fork | CreateProcess | Create a new process |
| waitpid | WaitForSingleObject | Can wait for a process to exit |
| execve | (none) | CreateProcess = fork + execve |
| exit | ExitProcess | Terminate execution |
| open | CreateFile | Create a file or open an existing file |
| close | CloseHandle | Close a file |
| read | ReadFile | Read data from a file |
| write | WriteFile | Write data to a file |
| lseek | SetFilePointer | Move the file pointer |
| stat | GetFileAttributesEx | Get various file attributes |
| mkdir | CreateDirectory | Create a new directory |
| rmdir | RemoveDirectory | Remove an empty directory |
| link | (none) | Win32 does not support links |
| unlink | DeleteFile | Destroy an existing file |
| mount | (none) | Win32 does not support mount |
| umount | (none) | Win32 does not support mount |
| chdir | SetCurrentDirectory | Change the current working directory |
| chmod | (none) | Win32 does not support security (although NT does) |
| kill | (none) | Win32 does not support signals |
| time | GetLocalTime | Get the current time |

## The Shell

- In computing, a shell is a piece of software that provides an interface for users and provides access to the services of a kernel.
- Operating system shells generally fall into one of two categories: command-line and graphical.
- Command-line shells provide a command-line interface (CLI) to the operating system, while graphical shells provide a graphical user interface (GUI).
- In either category the primary purpose of the shell is to invoke or "launch" another program; however, shells frequently have additional capabilities such as viewing the contents of directories.

### Unix Shell

Many shells exist in Unix, including

- Bourne shell (sh)
- Almquist shell (ash)
- Debian Almquist shell (dash)
- Bourne-Again shell (bash)
- Korn shell (ksh)
- Z shell (zsh)
- C shell (csh)
- Stand-alone Shell (sash)

All of them support the functionality described below, which derives from the original shell (*sh*).

When any user logs in, a shell is started up. It starts out by typing the **prompt,** a character such as a dollar sign, which tell the user that the shell is waiting to accept a command.

If the user types
*date*
This shell creates a child process and runs the *date* program as the child. While the child process is running, the shell waits for it to terminate. When the child finishes, the shell types the prompt again and tries to read the next input line.

date>file (the standard output is redirected to a file)

sort <file1>file2 (invokes the sort program with input take from file1 and output sent to file2)

cat file1 file2 file3 | sort>/dev/lp ( invokes the cat program to concatenate three files and send the output to sort to arrange all the lines in alphabetical order). The output of sort is redirected to the file /dev/lp, typically the printer.

## Operating System Structure

### Monolithic Systems

- The OS in it is written as a collection of procedures, each of which can call any of the other ones whenever it needs to.
- This structure is subtitled "The Big Mess". The structure is that there is no structure.
- In this technique each procedure is free to call any other one, if the latter provides some useful computation than the former needs.
- To construct the actual object of the operating system when this approach is used, one first compiles all the individuals' procedures, or files containing the procedures and then binds them all together into a single object file using the system linker.
- In terms of information hiding, there is essentially none – every procedure is visible to every other procedure.

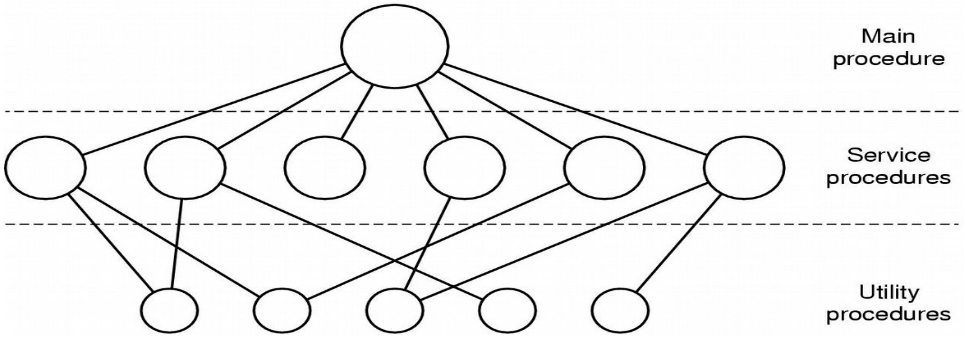
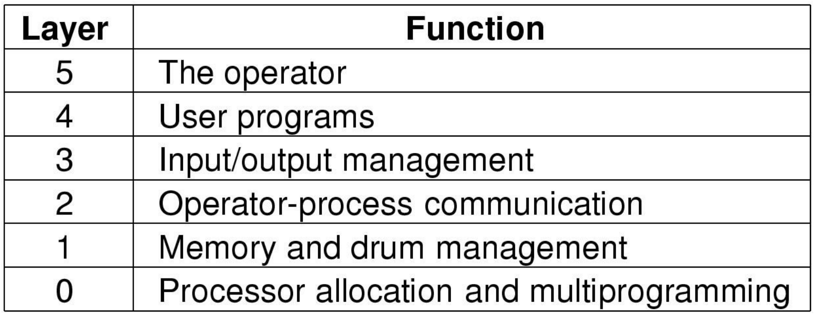


**Fig: A simple structuring model for a monolithic system**

1. The main program that invokes the requested service procedure.
2. A set of service procedures that carry out the system calls.
3. A set of utility procedures that help the service procedures.

### Layered Systems

- To generalize the monolithic system into the hierarchy of layers
- This system had 6 layers.

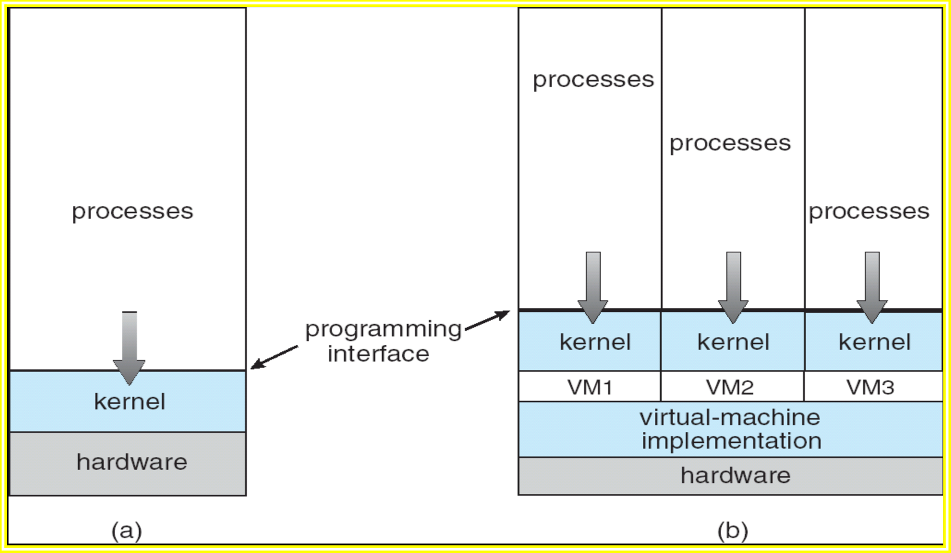


| Layer | Function |
|---|---|
| 5 | The operator |
| 4 | User programs |
| 3 | Input/output management |
| 2 | Operator-process communication |
| 1 | Memory and drum management |
| 0 | Processor allocation and multiprogramming |

**Fig: Structure of the operating system**

### Virtual Machines

- A *virtual machine* takes the layered approach to its logical conclusion. It treats hardware and the operating system kernel as though they were all hardware.
- The operating system creates the illusion of multiple processes, each executing on its own processor with its own (virtual) memory.
- The resources of the physical computer are shared to create the virtual machines
- CPU scheduling can create the appearance that users have their own processor
- Spooling and a file system can provide virtual card readers and virtual line printers
- A normal user time-sharing terminal serves as the virtual machine operator's console



(a) Non virtual machine (b) virtual machine

- The virtual-machine concept provides complete protection of system resources since each virtual machine is isolated from all other virtual machines. This isolation, however, permits no direct sharing of resources.
- A virtual-machine system is a perfect vehicle for operating-systems research and development. System development is done on the virtual machine, instead of on a physical machine and so does not disrupt normal system operation.
- The virtual machine concept is difficult to implement due to the effort required to provide an *exact* duplicate to the underlying machine