Computer System Architecture - It is generally classified based on the number of general purpose processors that are used.

1. Single Processor
2. Multi processor
3. Clustered

1. **Single Processor System** - **If there is only one general-purpose CPU, then the system is a single-processor system.**

- There is one main CPU capable of executing a general-purpose instruction set, including instructions from user processes.
- Almost all single processor systems have other special-purpose processors as well.
- Eg. device-specific processors, such as disk, keyboard, and graphics controllers , I/O processors that move data rapidly among the components of the system.
- All of these special-purpose processors run a limited instruction set and do not run user processes.
- Sometimes, they are managed by the operating system, in that the operating system sends them information about their next task and monitors their status.
- For example, a disk-controller microprocessor receives a sequence of requests from the main CPU and implements its own disk queue and scheduling algorithm. This arrangement relieves the main CPU of the overhead of disk scheduling.
- PCs contain a microprocessor in the keyboard to convert the keystrokes into codes to be sent to the CPU.
- In other systems or circumstances, special-purpose processors are low-level components built into the hardware. The operating system cannot communicate with these processors; they do their jobs autonomously.

2. Multi processor system - **If there are multiple general-purpose CPU, then the system is a multi-processor system.**

Such systems have two or more processors in close communication, sharing the computer bus and sometimes the clock, memory, and peripheral devices.

Key advantages :

I. **Increased throughput.**

- By increasing the number of processors, we expect to get more work done in less time.
- The speed-up ratio with N processors is not N because a certain amount of overhead is incurred in keeping all the parts working correctly.
- This overhead, plus contention for shared resources, lowers the expected gain from additional processors.

2. **Economy of scale.**

- Multiprocessor systems can cost less than equivalent multiple single-processor systems, because they can share peripherals, mass storage, and power supplies.

- If several programs operate on the same set of data, it is cheaper to store those data on one disk and to have all the processors share them than to have many computers with local disks and many copies of the data.

3. **Increased reliability.**
   - If functions can be distributed properly among several processors, then the failure of one processor will not halt the system, only slow it down.
   - If we have ten processors and one fails, then each of the remaining nine processors can pick up a share of the work of the failed processor. Thus, the entire system runs only 10 percent slower, rather than failing altogether.

Graceful Degradation - The ability to continue providing service proportional to the level of surviving hardware is called graceful degradation.
Fault tolerant - Systems that can suffer a failure of any single component and still continue operation. Fault tolerance requires a mechanism to allow the failure to be detected, diagnosed, and, if possible, corrected.

Types of Multiprocessing

a) **Asymmetric multiprocessing**, in which each processor is assigned a specific task. A **boss** processor controls the system; the other processors either look to the boss for instruction or have predefined tasks. This scheme defines a boss–worker relationship. The boss processor schedules and allocates work to the worker processors.

b) Symmetric Multiprocessing(SMP) - in which each processor performs all tasks within the operating system. SMP means that all processors are peers; no boss–worker relationship exists between processors.

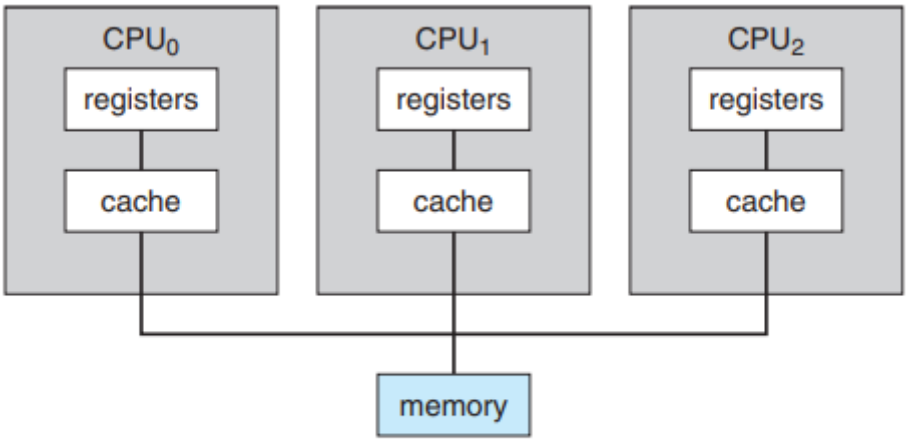


Figure 1.6 Symmetric multiprocessing architecture.

Features of multiprocessing :
- Multiprocessing adds CPUs to increase computing power.
- If the CPU has an integrated memory controller, then adding CPUs can also increase the amount of memory addressable in the system.

- Either way, multiprocessing can cause a system to change its memory access model from uniform memory access (UMA) to non-uniform memory access (NUMA).
- UMA is defined as the situation in which access to any RAM from any CPU takes the same amount of time. With NUMA, some parts of memory may take longer to access than other parts, creating a performance penalty.

Dual Core Systems/ Multi core - Multiple computing cores on a single chip.

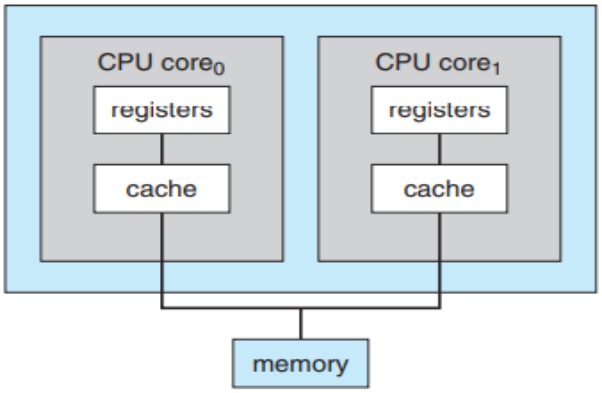


Figure 1.7 A dual-core design with two cores placed on the same chip.

- They can be more efficient than multiple chips with single cores because on-chip communication is faster than between-chip communication.
- In addition, one chip with multiple cores uses significantly less power than multiple single-core chips.
- It is important to note that while multicore systems are multiprocessor systems, not all multiprocessor systems are multicore.

3. **Clustered Systems** - Combining multiple CPU's together

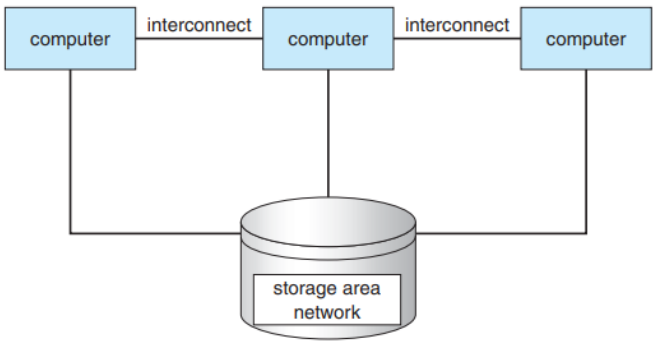


Figure 1.8 General structure of a clustered system.

Features of Clustered System :

- It is composed of two or more individual systems—or nodes—joined together. Such systems are considered loosely coupled.
- Each node may be a single processor system or a multicore system.
- It shares storage and are closely linked via a local-area network LAN.
- It is used to provide high-availability service—that is, service will continue even if one or more systems in the cluster fail.
- Generally, we obtain high availability by adding a level of redundancy in the system.

How Clustering helps?

- A layer of cluster software runs on the cluster nodes.
- Each node can monitor one or more of the others (over the LAN).
- If the monitored machine fails, the monitoring machine can take ownership of its storage and restart the applications that were running on the failed machine.
- The users and clients of the applications see only a brief interruption of service.

Types

I. Asymmetric Clustering

- one machine is in hot-standby mode while the other is running the applications.
- The hot-standby host machine does nothing but monitor the active server. If that server fails, the hot-standby host becomes the active server.

II. **Symmetric Clustering**

- In symmetric clustering, two or more hosts are running applications and are monitoring each other.
- This structure is obviously more efficient, as it uses all of the available hardware. However it does require that more than one application be available to run.

Parallelization:

- divides a program into separate components that run in parallel on individual computers in the cluster.
- Typically, these applications are designed so that once each computing node in the cluster has solved its portion of the problem, the results from all the nodes are combined into a final solution.
- Parallel clusters allow multiple hosts to access the same data on shared storage. Because most operating systems lack support for simultaneous data access by multiple hosts, parallel clusters usually require the use of special versions of software and special releases of applications

Distributed lock manager - Functionality used to ensure that there is no resource conflict when multiple hosts access the same resource.

Storage Area Networks

- It allows many systems to attach to a pool of storage.
- If the applications and their data are stored on the SAN, then the cluster software can assign the application to run on any host that is attached to the SAN.
- If the host fails, then any other host can take over. In a database cluster, dozens of hosts can share the same database, greatly increasing performance and reliability.

MULTIPROGRAMMING VS TIME SHARING SYSTEM

Need of Multiprogramming

A single program cannot, in general, keep either the CPU or the I/O devices busy at all times. Single users frequently have multiple programs running. Multiprogramming increases CPU utilization by organizing jobs (code and data) so that the CPU always has one to execute.

How multiprogramming works:

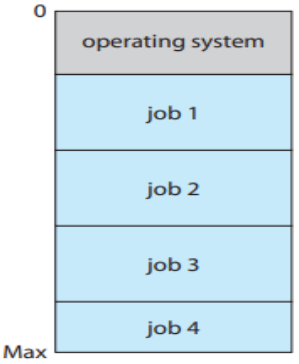


Figure 1.9 Memory layout for a multiprogramming system.

- The operating system keeps several jobs in memory simultaneously (Figure 1.9). Since, in general, main memory is too small to accommodate 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 main memory.
- Job scheduler takes a set of jobs from this pool and brings it into memory based on scheduling algorithms.
- Multiprogrammed systems provide an environment in which the various system resources (for example, CPU, memory, and peripheral devices) are utilized effectively, but they do not provide for user interaction with the computer system.
- When a process has to wait (for I/O for example), OS switches to another job.

Timesharing System - is a logical extension of multiprogramming. In time-sharing systems, the CPU executes multiple jobs by switching among them, but the switches occur so frequently that the users can interact with each program while it is running.

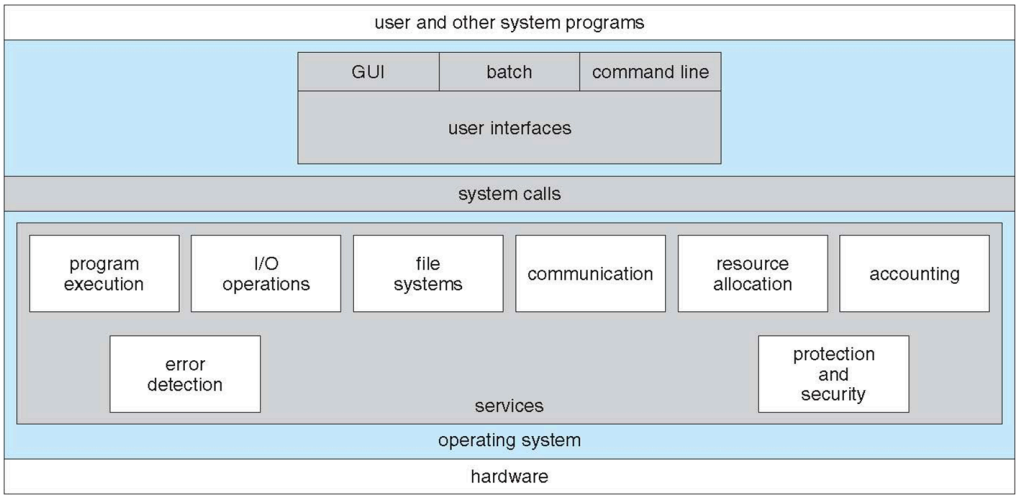
- Time sharing requires an interactive computer system, which provides direct communication between the user and the system. The user gives instructions to the operating system or to a program directly, using a input device such as

a keyboard, mouse, touch pad, or touch screen, and waits for immediate results on an output device.

- response time should be short—typically less than one second.
- A time-shared operating system allows many users to share the computer simultaneously. Since each action or command in a time-shared system tends to be short, only a little CPU time is needed for each user. As the system switches rapidly 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.
- A time-shared operating system uses CPU scheduling and multiprogramming to provide each user with a small portion of a time-shared computer. Each user has at least one separate program in memory

S.No.	TIME SHARING	MULTIPROGRAMMING
01.	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.	Multiprogramming operating system allows to execute multiple processes by monitoring their process states and switching in between processes.
02.	Processors time is shared with multiple users that's why it is called as time sharing operating system.	Processor and memory underutilization problem is resolved and multiple programs runs on CPU that's why it is called multiprogramming.
03.	In this process, two or more users can use a processor in their terminal.	In this, the process can be executed by a single processor.
04.	Time sharing OS has fixed time slice.	Multi-programming OS has no fixed time slice.
05.	In time sharing OS system, execution power is taken off before finishing of execution.	In multi-programming OS system before finishing a task the execution power is not taken off.
06.	Here the system works for the same or less time on each processes.	Here the system does not take same time to work on different processes.
07.	In time sharing OS system depends on time to switch between different processes.	In Multiprogramming OS, system depends on devices to switch between tasks such I/O interrupts etc.
08.	System model of time sharing system is multiple programs and multiple users.	System model of multiprogramming system is multiple programs.
09.	Time sharing system minimizes response time.	Multiprogramming system maximizes processor use.

Operating-System Services



1. **User interface**
    a. Almost all operating systems have a user interface (UI). This interface can take several forms.
        i. Eg.1 command-line interface (CLI), which uses text commands and a method for entering them (say, a keyboard for typing in commands in a specific format with specific options).
        ii. Eg 2 batch interface, in which commands and directives to control those commands are entered into files, and those files are executed.
        iii. Eg 3 a graphical user interface (GUI) is used.
2. **Program execution.**
    a. 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 (indicating error).
3. **I/O operations**
    a. A running program may require I/O, which may involve a file or an I/O device. For specific devices, special functions may be desired (such as recording to a CD or DVD drive or blanking a display screen).
    b. For efficiency and protection, users usually cannot control I/O devices directly. Therefore, the operating system must provide a means to do I/O.
4. **File-system manipulation**
    a. The file system is of particular interest. Obviously, programs need to read and write files and directories. They also need to create and delete them by name, search for a given file, and list file information.
    b. Finally, some operating systems include permissions management to allow or deny access to files or directories based on file ownership.
5. **Communications**
    a. There are many circumstances in which one process needs to exchange information with another process. Such communication may occur between processes that are executing on the same computer or between processes that are executing on different computer systems tied together by a computer network.

b. Communications may be implemented via shared memory, in which two or more processes read and write to a shared section of memory, or message passing, in which packets of information in predefined formats are moved between processes by the operating system.

6. **Error detection**
    a. The operating system needs to be detecting and correcting errors constantly.
    b. 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 disk, a connection failure on a network, or lack of paper in the printer), and in the user program (such as an arithmetic overflow, an attempt to access an illegal memory location, or a too-great use of CPU time).
    c. For each type of error, the operating system should take the appropriate action to ensure correct and consistent computing. Sometimes, it has no choice but to halt the system. At other times, it might terminate an error-causing process or return an error code to a process for the process to detect and possibly correct.

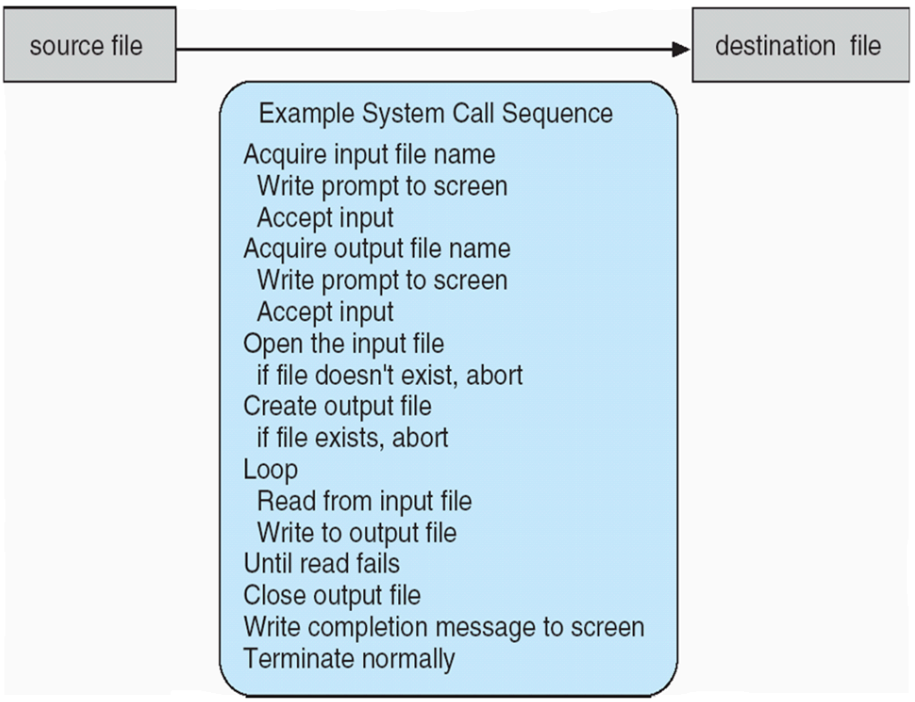
7. **Resource allocation**
    a. When there are multiple users or multiple jobs running at the same time, resources must be allocated to each of them.
    b. The operating system manages many different types of resources such as CPU cycles, main memory, and file storage.
    c. For instance, in determining how best to use the CPU, operating systems have CPU-scheduling routines that take into account the speed of the CPU, the jobs that must be executed, the number of registers available, and other factors. There may also be routines to allocate printers, USB storage drives, and other peripheral devices.
8. **Accounting**
    a. We want to keep track of which users use how much and what kinds of computer resources.
    b. This record keeping may be used for accounting (so that users can be billed) or simply for accumulating usage statistics.
    c. Usage statistics may be a valuable tool for researchers who wish to reconfigure the system to improve computing services.
9. **Protection and security**
    a. The owners of information stored in a multiuser or networked computer system may want to control use of that information.
    b. When several separate processes execute concurrently, it should not be possible for one process to interfere with the others or with the operating system itself.
    c. Protection involves ensuring that all access to system resources is controlled.
    d. Security of the system from outsiders is also important. Such security starts with requiring each user to authenticate himself or herself to the system, usually by means of a password, to gain access to system resources.

System Calls

- A system call is a routine that allows a user application to request actions that require special privileges.

- A system call is a method for a computer program to request a service from the kernel of the operating system on which it is running. A system call is a method of interacting with the operating system via programs. A system call is a request from computer software to an operating system's kernel.

- A series of sequence calls is required to perform file copy from one file to another file.



Frequently, systems execute thousands of system calls per second. Most programmers never see this level of detail, however. Typically, application developers design programs according to an application programming interface (API).
The API specifies a set of functions that are available to an application programmer, including the parameters that are passed to each function and the return values the programmer can expect.

EXAMPLE OF STANDARD API

As an example of a standard API, consider the `read()` function that is available in UNIX and Linux systems. The API for this function is obtained from the `man` page by invoking the command

```
man read
```

on the command line. A description of this API appears below:

```
#include <unistd.h>

ssize_t     read(int fd, void *buf, size_t count)
```

return value — function name — parameters

A program that uses the `read()` function must include the `unistd.h` header file, as this file defines the `ssize_t` and `size_t` data types (among other things). The parameters passed to `read()` are as follows:

- `int fd`—the file descriptor to be read
- `void *buf`—a buffer where the data will be read into
- `size_t count`—the maximum number of bytes to be read into the buffer

On a successful read, the number of bytes read is returned. A return value of 0 indicates end of file. If an error occurs, `read()` returns −1.

Working of API with system calls:

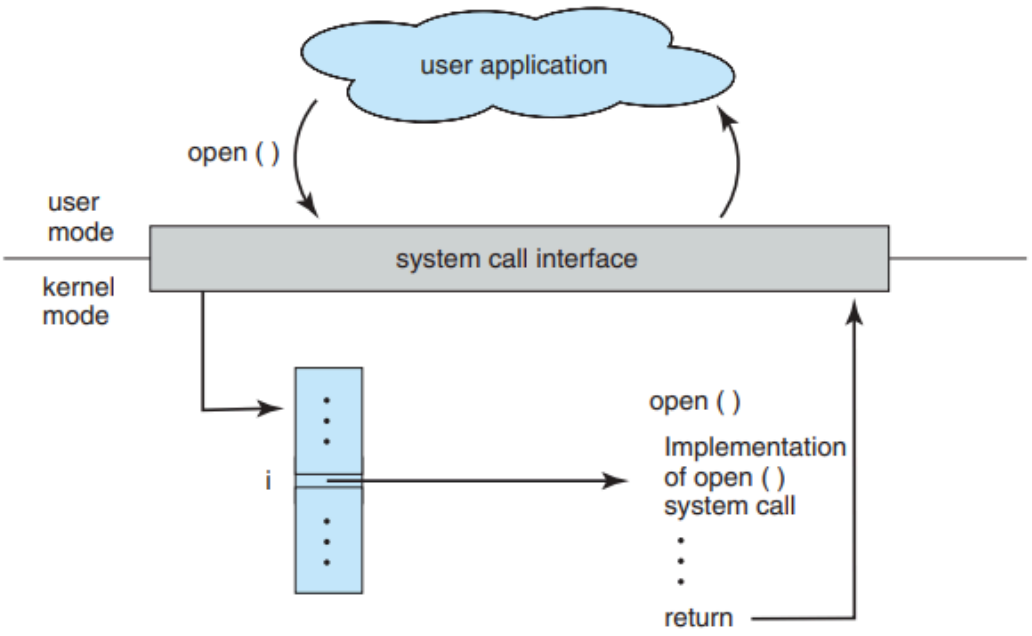


Figure 2.6 The handling of a user application invoking the `open()` system call.

- System call interface serves as the link to system calls made available by the operating system.
- The system-call interface intercepts function calls in the API and invokes the necessary system calls within the operating system.
- Typically, a number is associated with each system call, and the system-call interface maintains a table indexed according to these numbers.
- The interface then invokes the intended system call in the operating-system kernel and returns the status of the system call and any return values.
- The caller need not know about how the system call is implemented or what it does during execution.
- Rather, the caller need only obey the API and understand what the operating system will do as a result of the execution of that system call.
- Thus, most of the details of the operating-system interface are hidden from the programmer by the API and are managed by the run-time support library.

Passing of Parameters to System Calls:

- Three general methods used to pass parameters to the OS
  - Simplest: pass the parameters in registers
- In some cases, may be more parameters than registers
  - Parameters stored in a block, or table, in memory, and address of block passed as a parameter in a register
  -
- Parameters placed, or pushed, onto the stack by the program and popped off the stack by the operating system
- Block and stack methods do not limit the number or length of parameters being passed

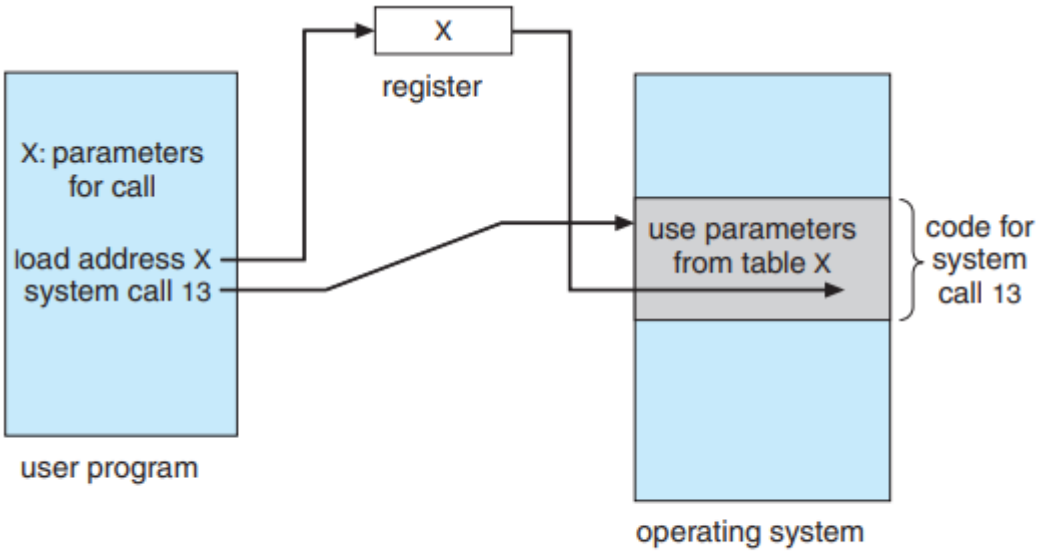


Figure 2.7 Passing of parameters as a table.

Types of System Calls

- Process control
  - end, abort
  - load, execute
  - create process, terminate process
  - get process attributes, set process attributes
  - wait for time
  - wait event, signal event
  - allocate and free memory
- File management
  - create file, delete file
  - open, close
  - read, write, reposition
  - get file attributes, set file attributes
- Device management
  - request device, release device
  - read, write, reposition
  - get device attributes, set device attributes
  - logically attach or detach devices
- Information maintenance
  - get time or date, set time or date
  - get system data, set system data
  - get process, file, or device attributes
  - set process, file, or device attributes
- Communications
  - create, delete communication connection
  - send, receive messages
  - transfer status information
  - attach or detach remote devices

EXAMPLES OF WINDOWS AND UNIX SYSTEM CALLS

	Windows	Unix
Process Control	`CreateProcess()`	`fork()`
	`ExitProcess()`	`exit()`
	`WaitForSingleObject()`	`wait()`
File Manipulation	`CreateFile()`	`open()`
	`ReadFile()`	`read()`
	`WriteFile()`	`write()`
	`CloseHandle()`	`close()`
Device Manipulation	`SetConsoleMode()`	`ioctl()`
	`ReadConsole()`	`read()`
	`WriteConsole()`	write()
Information Maintenance	`GetCurrentProcessID()`	`getpid()`
	`SetTimer()`	`alarm()`
	`Sleep()`	`sleep()`
Communication	`CreatePipe()`	`pipe()`
	`CreateFileMapping()`	`shm_open()`
	`MapViewOfFile()`	`mmap()`
Protection	`SetFileSecurity()`	`chmod()`
	`InitlializeSecurityDescriptor()`	`umask()`
	`SetSecurityDescriptorGroup()`	`chown()`

Important System Calls Used in OS

1. wait()

In certain systems, a process must wait for another process to finish running before continuing. When a parent process creates a child process, the parent process's execution is suspended until the child process has finished running.

With a wait() system call, the parent process is automatically suspended. Control returns to the parent process once the child process has completed its execution.

2. fork()

The fork system call in OS is used by processes to make copies of themselves. By using this system, the Call parent process can create a child process, and the parent process's execution will be halted while the child process runs.

3. exec()

When an executable file replaces an earlier executable file within the context of an active process, this system call is executed. The original process identifier is still present even though a new process is not created; instead, the new process replaces the old one's stack, data, head, data, etc.

4. kill()

The OS uses the kill() system call to urge processes to terminate by sending them a signal. A kill system call can have different meanings and is not always used to end a process.

5. exit()

An exit system call is used when the programme must be stopped. When the exit system call is used, the resources that the process was using were released.

System Programs

- provide a convenient environment for program development and execution. Some of them are simply user interfaces to system calls.

- **File management**. These programs create, delete, copy, rename, print, dump, list, and generally manipulate files and directories.
- **Status information**. Some programs simply ask the system for the date, time, amount of available memory or disk space, number of users, or similar status information. Others are more complex, providing detailed performance, logging, and debugging information. Typically, these programs format and print the output to the terminal or other output devices or files or display it in a window of the GUI. Some systems also support a registry, which is used to store and retrieve configuration information.
- **File modification**. Several text editors may be available to create and modify the content of files stored on disk or other storage devices. There may also be special commands to search contents of files or perform transformations of the text.
- **Programming-language support**. Compilers, assemblers, debuggers, and interpreters for common programming languages (such as C, C++, Java, and PERL) are often provided with the operating system or available as a separate download.
- **Program loading and execution**. Once a program is assembled or compiled, it must be loaded into memory to be executed. The system may provide absolute loaders, relocatable loaders, linkage editors, and overlay loaders. Debugging systems for either higher-level languages or machine language are needed as well.
- **Communications**. These programs provide the mechanism for creating virtual connections among processes, users, and computer systems. They allow users to send messages to one another's screens, to browse Web pages, to send e-mail messages, to log in remotely, or to transfer files from one machine to another.
- **Background services**. All general-purpose systems have methods for launching certain system-program processes at boot time. Some of these processes terminate after completing their tasks, while others continue to run until the system is halted. Constantly running system-program processes are known as services, subsystems, or daemons. ~~One example is~~

Application Programs

- It includes Web browsers, word processors and text formatters, spreadsheets, database systems, compilers, plotting and statistical-analysis packages, and games.
- Don't pertain to system
- Run by users
- Not typically considered part of OS
- Launched by command line, mouse click, finger poke