Introduction

I . Features of Unix OS

SHELL :

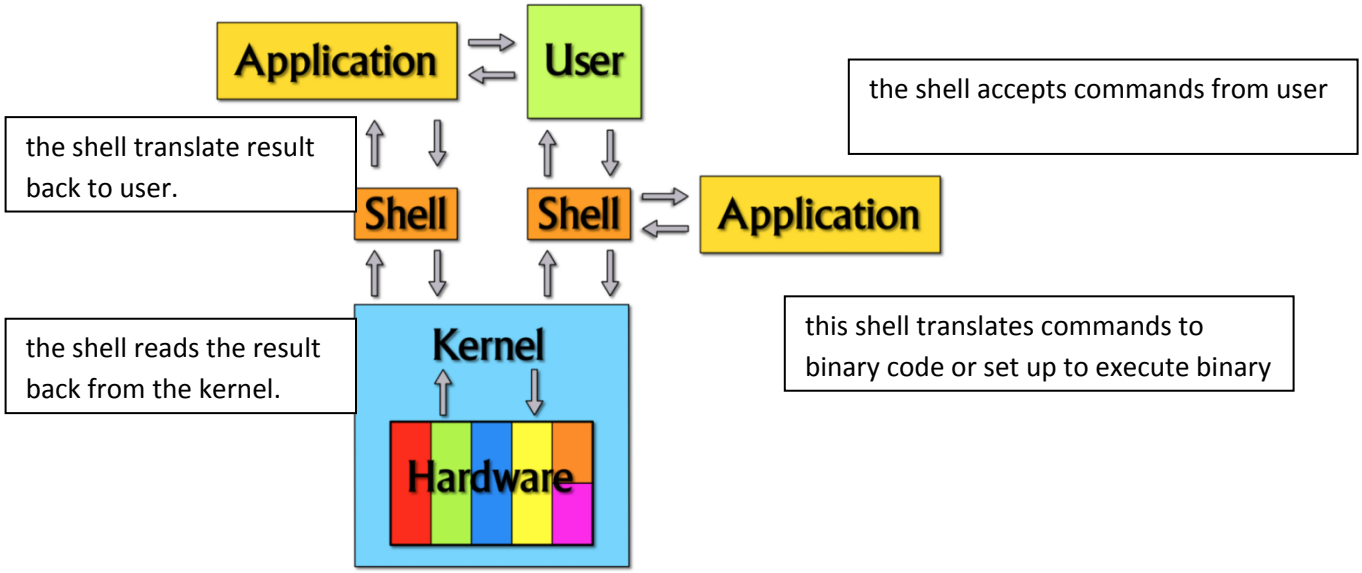
A **Shell** provides you with an interface to the Unix system and kernel.

It gathers input from you and executes programs based on that input. When a program finishes executing, it displays that program's output.

Shell is an environment in which we can run our commands, programs, and shell scripts.

There are different flavors of a shell, just as there are different flavors of operating systems. Each flavor of shell has its own set of recognized commands and functions.

Shell is also known as commands.



Shell Types:

In UNIX there are two major types of shells:

1. Bourne shell
2. c shell

1. **The Bourne shell :** If you are using a Bourne-type shell, the default prompt is the $ character.
2. **The C shell:** If you are using a C-type shell, the default prompt is the % character.

There are again various subcategories for **Bourne Shell** which are listed as follows:

- **Bourne shell (sh):** The original UNIX shell was written in the mid-1970s by Stephen R. Bourne while he was at AT&T Bell Labs in New Jersey.

  It is most widely used shell in unix world.

  The Bourne shell was the first shell to appear on UNIX systems, thus it is referred to as "the shell".

  The Bourne shell is usually installed as **/bin/sh** on most versions of UNIX. For this reason, it is the shell of choice for writing scripts to use on several different versions of UNIX.

  It provides **$ prompt** on unix installation as trademark of bourne shell.

- **Korn shell (ksh):** korn shell is the unix shell developed by David korn of Bell labs.

  Is is considered as the family member of Bourne shell as it uses the $ symbol of Bourne shell.

  It is also names as ***ksh*** programmatic ally and it most widely used shell.

  it can completely replaced bourne shell in a system and has more functions that are built into shell making it more efficient.

- **Bourne Again shell (bash):** It is the free version of Bourne shell and comes with all UNIX/Linux systems as free with some additional features like command line editing.

  Its program name is ***bash***. It can read commands from file called scripts.

  Like all Unix shells it supports the following:

  - File name wildcarding
  - Piping
  - Hear documents
  - Command execution
  - Variables and control structures for condition testing and iteration.

- POSIX shell (sh):

The different C-type shells follow:

- C shell (csh) :

  Is a UNIX enhancement written by Bill Joy at the University of California at Berkeley.

  C shell along with Bourne and Korn, are there most popular and commonly used shells. **csh is the program name for C shell.**

  Incorporated features for interactive use, such as aliases and command history.

Includes convenient programming features, such as built-in arithmetic and a C-like expression syntax.

For the C shell the:
Command full-path name is **/bin/csh**.
Non-root user default prompt is **hostname %.**
Root user default prompt is hostname **#**.

- **TENEX/TOPS C shell (tcsh):** it pronounced as tee-cee shell. it is a compatible version of the C shell. it is used in linux environment.
- **pdksh:** It stands for Public Domain Korn Shell. Linux offers pdksh as a substitute of ksh shell.

Shell Features:

1. Interactive environment: the shell allows user to create a dialogue, i.e. communication channel between user and the host unix system. this dialogue terminates until the user ends the session.
2. shell scripts: A shell script is small computer program that is designed to be run or executed by the Unix shell, which is a command-line interpreter. A shell script is basically a set of commands that the shell in a Unix-based operating system follows. Like actual programs, the commands in the shell script can contain parameters and subcommands that tell the shell what to do. The shell script is usually contained in a simple text file.
3. Input/output redirection: it is a function of shell that redirects the output from program to a destination other than the screen. this way, user can save output from command into a file and redirect it into a printer, another terminal on the network or even other program. similarly, a shell can make a program that accepts input from other than the keyboard by redirecting its input from other source.
4. Piping Mechanism: Piping connects the output of one process to the input of a second. Piping uses the symbol "|". A filter is a process which is between two pipes. It simply changes the information coming down the pipe.
   Standard input for a process can be drawn from the script file itself. This uses the special redirection symbol "<<". The name which follows the redirection symbol is a tag for the end of the input. That is, the contents of the script file will be read as standard input to the process until a input line matching the name following the redirection symbol.
5. Metacharacter Facilities/filename substitution: shell recognize the *,? or [..] as special characters when reading the argument from a command line. then shell perform file name expansion on this list before executing the request program.

Ex: ls s* -- it substitutes all the file names

6. Background Processing: A multitasking facility allows the user to run command in the background. This allows the command to be processed while the user can proceed with other tasks. when background task is completed, user is notified.
7. Customize Environment: the shell is your working environment. facilities are available by which the shell can be customized for your personal need.
8. programming language construct: the shell includes features that allow it to be used as programming language. these feature can be used to build shell scripts that perform complex operation.
9. shell variables: the user can control the behavior of shell as well as other programs and utilities, by storing data in variables.

Kernel:

Unix system has 3 levels: **user, kernel and hardware**.

A kernel can be contrasted with a shell, the outermost part of an operating system that interacts with user commands.

A kernel is the core component of an operating system.

Using inter process communication and system calls, it acts as a bridge between applications and the data processing performed at the hardware level.

When an operating system is loaded into memory, the kernel loads first and remains in memory until the operating system is shut down again.

The kernel is responsible for low-level tasks such as disk management, task management and memory management.

Block diagram of System kernel:

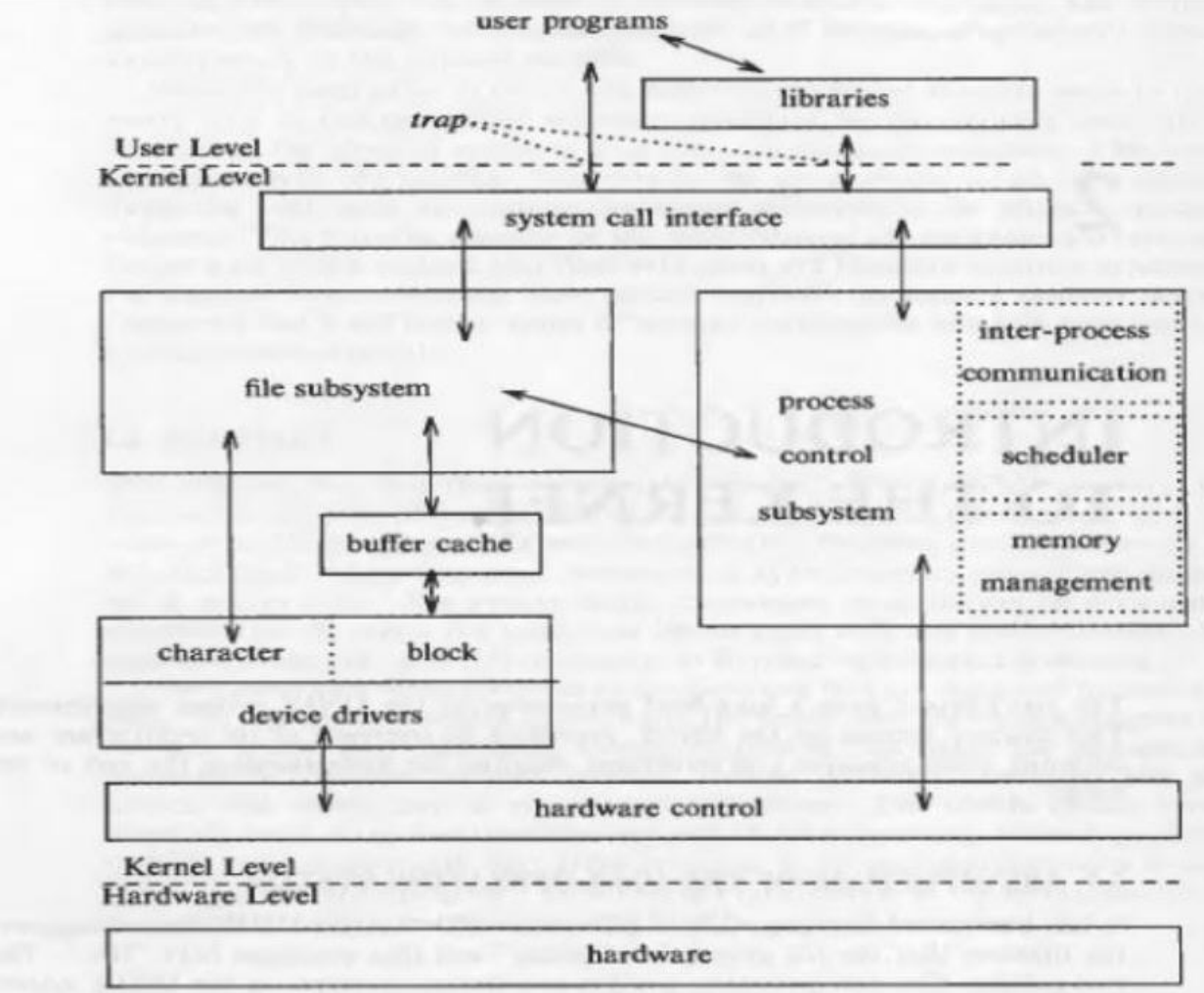


Figure 2.1. Block Diagram of the System Kernel

Above Figure gives a block diagram of the kernel, showing various modules and their relationships to each other.

In particular, it shows the file subsystem on the left and the process control subsystem on the right, the two major component of the kernel.

The diagram serves as a useful logical view of the kernel, although in practice the kernel deviates from the model because some modules interact with the internal operations of others.

Figure shows three levels: user, kernel, and hardware.

The system call and library interface represent the border between user programs and the kernel.

System calls look like ordinary function calls in C programs, and libraries map these function calls to the primitives needed to enter the operating system.

Assembly language programs may invoke system calls directly without a system call library, however. Programs frequently use other libraries such as the standard I/O library to provide a more sophisticated use of the system calls. The libraries are linked with the programs at compile time.

The figure partitions the set of system calls into those that interact with the file subsystem and those that interact with the process control subsystem. The file subsystem manages files, allocating file space, administering free space, controlling access to files, and retrieving data for users.

Processes interact with the file subsystem via a specific set of system calls, such as open (to open a file for reading or writing), close, read, write, stat (query the attributes of a file).

The file subsystem accesses file data using a buffering mechanism that regulates data flow between the kernel and secondary storage devices.

Device drivers are the kernel modules that control the operation of peripheral devices. Block I/O devices are random access storage devices alternatively, their device drivers make them appear to be random access storage devices to the rest of the system.

The process control subsystem is responsible for process synchronization, interprocess communication, memory management, and process scheduling.

The memory management module controls the allocation of memory.

The scheduler module allocates the CPU to processes. It schedules them to run in turn until they voluntarily relinquish the CPU while awaiting a resource or until the kernel preempts them when their recent run time exceeds a time quantum.