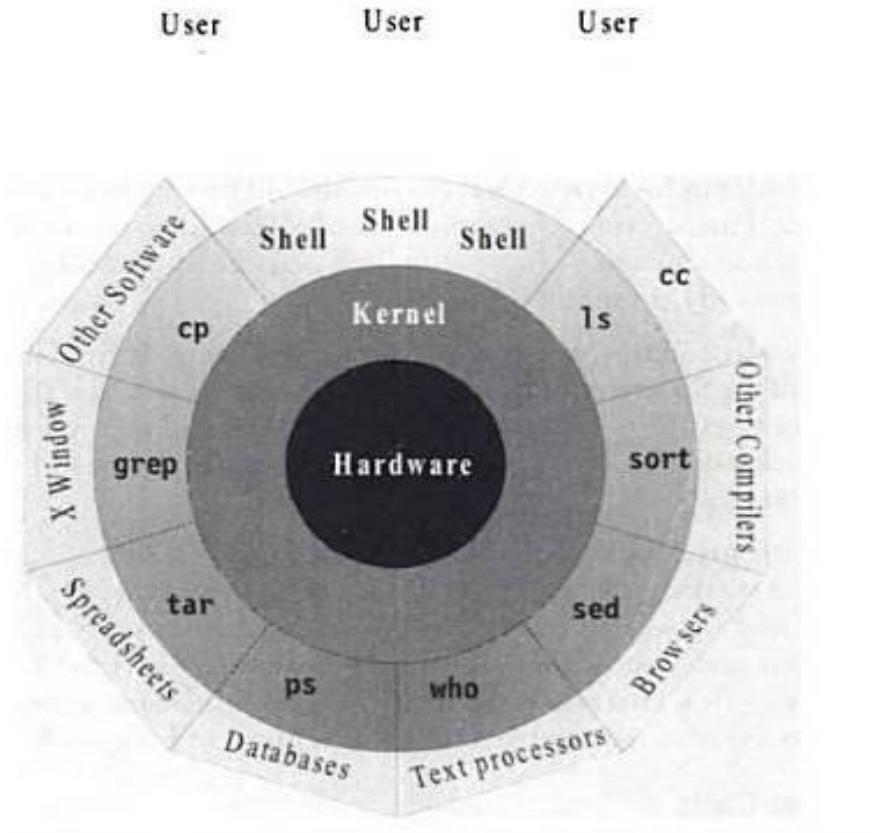
MODULE-1

UNIX Components/ Architecture(S/W)

- Division of Labor: Kernel and Shell
- The File and Process
- The System Calls

UNIX Components/ Architecture(S/W) Division of Labor

The Kernel-Shell Relationship is shown in below:



UNIX Components/ Architecture(S/W) Division of Labor

- The kernel interacts with the m/c's H/W, and the shell with the user.
- The Kernel is the core of the OS – a collection of routines mostly written in C.
  - Is loaded into memory when the system is booted and communicates directly with the h/w.
  - The user programs that need to access the h/w use the service of the kernel.
  - User programs access kernel through a set of functions called SYSTEM CALLS.

UNIX Components/ Architecture(S/W) Division of Labor

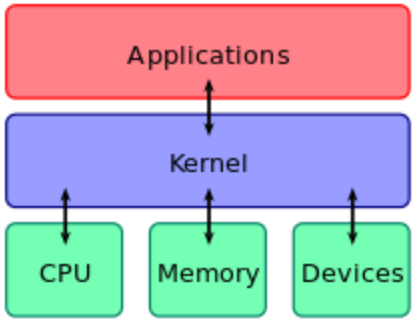
  - It manages the system's memory
  - Schedule processes
    - Decides their priorities
  - Performs other important tasks.
- It is often called the OS - a program's gateway to the computer's resources.

UNIX Components/ Architecture(S/W) Division of Labor

- UNIX Shell is an interface between the user and the kernel.
- UNIX shell is a command-line interpreter or a command-line user interface.
  - When a user enter a command using the KB at the command prompt, the shell thoroughly examines the KB i/p for a special characters.
  - Upon finding a special character, it rebuilds a simplified command line, and finally communicates with the kernel to see that the command is executed.

Note:

- Functions of the kernel
  - Memory management
  - Device management
  - System calls



Note:

- The **kernel** is a computer program that is the core of a computer's operating system, with complete control over everything in the system.
- It is the first program loaded on start-up.
- It handles the rest of start-up as well as input/output requests from software, translating them into data-processing instructions for the central processing unit.
- It handles memory and peripherals like keyboards, monitors, printers, and speakers.

Note:

- Under Unix, the operating system consists of many utilities along with the master control program, the kernel.
  - The kernel provides services to start and stop programs, handles the file system and other common "low-level" tasks that most programs share, and
  - schedules access to avoid conflicts when programs try to access the same resource or device simultaneously. To mediate such access, the kernel has special rights, reflected in the division between user space and kernel space.

UNIX Components/ Architecture(S/W)
The File and Process

- Two simple entities support the UNIX system- the file and process.
- Files have places and processes have life.

FILE:

- A File is just an array of bytes and can contain virtually anything.
- The Unix file system is organized around a **single structure of directories (Inverted Tree Structure)**.
- Files can be moved from one directory to another.

UNIX Components/ Architecture(S/W) The File and Process

- UNIX treats directories and devices as members of the file system.
- The dominant file type is text, and the behavior of the system is mainly controlled by text files.
- UNIX provides many text manipulation tools that can edit files without using an editor.

UNIX Components/ Architecture(S/W) Note:

- The Unix file system is organized around a **single structure of directories**, where each directory can contain more directories (often called subdirectories) and/or files.
- The entire file system, often spanning many machines and disks, can be visualized as a tree.
- Picture this tree as growing upside down, with the root at the top and the leaves toward the bottom. The leaves are all text and executable files, while the root, trunk, limbs, branches, and twigs are all directories.

UNIX Components/ Architecture(S/W) Note:

- The file system is called the directory tree, and the directory at the base of the tree is called the root directory.
- Every file and directory in the file system has a unique name, called its pathname. The pathname of the root directory is /.

UNIX Components/ Architecture(S/W) Note:

- As a Unix user, you are given control over one directory.
- This directory is called your home directory, and it is created when your account was established.
- This directory is your personal domain, over which you have complete control. You are free to create your own subtree of files and directories within your home directory.
- To determine the pathname of your home directory, enter the following command into a Unix shell window: cd; pwd

UNIX Components/ Architecture(S/W) The File and Process

PROCESS:

- A name given to a file when it is executed as a program.
- It is a “time image” of an executable file.
- Like files, processes also belong to a separate hierarchical tree structure.
- UNIX provides tools to control processes, move them between foreground and background, and even kill them.

UNIX Components/ Architecture(S/W) The System Calls

- System call sometimes referred to as a kernel call is the programmatic way in which a computer program requests a service from the kernel of the operating system it is executed on.
- This may include hardware-related services (for example, accessing a hard disk drive), creation and execution of new processes, and communication with integral kernel services such as process scheduling.
- System calls provide an essential interface between a process and the operating system.

Features of UNIX

1. **A Multiuser System**
2. **A Multitasking System**
3. **The Building-Block Approach**
4. **The UNIX Toolkit**
5. **Pattern Matching**
6. **Programming Facility**
7. **Documentation**

Features of UNIX

1. **A Multiuser System**

- The UNIX design allows multiple users to concurrently share hardware and software.
- Permits multiple programs to run and compete for the CPU.
  - Multiple users can run separate jobs.
  - A single user can also run multiple jobs.

Features of UNIX

2. <u>A Multitasking System</u>

- A single user can run multiple tasks concurrently.
  - E.g. A single user can edit a file, print another file, send email, browse the Web – all without leaving any applications.
- In multitasking environment, a user sees one job running in the foreground; the rest run in the background.

Features of UNIX

3. <u>The Building-Block Approach</u>

- The UNIX designers philosophy/ approach: **<u>"small is beautiful".</u>**
- Developed hundreds of commands each of which performed one simple job/task only.
- Commands can be **interconnected(using pipe |)** where one command o/p become an i/p to another command(**filter**).

Features of UNIX

4. <u>The UNIX Toolkit</u>

- With every new version of UNIX shipped there are general purpose tools, text manipulation utilities (called filters), compilers, interpreters, networked applications and system administrator tools.

Features of UNIX

5. <u>Pattern Matching</u>

- UNIX features very sophisticated pattern matching features.
- E.g. A user often need to search a file for a pattern, either to see the lines containing or not containing it or have it replaced with something else.
- **Tools – grep** and **sed**

Features of UNIX

6. <u>Programming Facility</u>

- UNIX provides facilities for implementing shell scripts – programs
  - that can invoke the UNIX commands
  - System's function can be controlled and automated
- UNIX provides necessary ingredients like control structures, loops, variables, Keywords.

Features of UNIX

7. <u>Documentation</u>

- Online – using man command.
- Internet
- Newsgroup
- Articles published in magazines and Journals
- Lecture notes

Other Important Features

1. **Portable:** UNIX can be installed on many hardware platforms.
2. **Networking:** Access to another system uses a standard communications protocol known as Transmission Control Protocol/Internet Protocol (TCP/IP).
3. **Organized File System:** UNIX has a very organized file and directory system that allows users to organize and maintain files.

Other Important Features

4. **Device Independence:** UNIX treats input/output devices like ordinary files. The source or destination for file input and output is easily controlled through a UNIX design feature called redirection.
5. **Utilities:** UNIX provides a rich library of utilities that can be use to increase user productivity.

Internal and External Commands

- **Built-in commands**: Commands that are part of shell and is executed by the shell.
  - E.gs bg, cd, echo, fg, printf etc.
- **External Commands**: External commands are those command which are stored as a separate binaries. Shell starts separate sub-process to execute them.
    - E.gs ls, dir, apropos, date, whatis etc.Binaries stored in /usr/bin or /bin

The type command

- *type* will display the command name's path. Possible command types are:
  - shell built-in
  - function
  - alias
  - hashed command
  - keyword
- The command returns a non-zero exit status if command names cannot be found.

Combining Commands

- Unix allows to specify more than one command in the command line.
- Each command has to be separated from the other by a semicolon.
- The semicolon is a known a metacharacter.
- E.g. wc note;ls -l note

Command: man

- **The man command**: an interface to the on-line reference manuals.
- **Display on screen**: Man displays one page (screen) at a time using a specific pager program: **more or less.**
  - **more** is a filter for paging through text one screenful at a time.
  - Less is a program similar to more , but which allows backward movement in the file as well as forward movement.

- **Navigation and Search**:
  - f or spacebar, advances the display by one screen of text at a time.
  - b moves back one screen.
  - Use forward slash(/) and the word to locate
- **Man documentation organization**:
  - EIGHT SECTIONS
    - NAME
    - SYNOPSIS
    - OPTIONS
    - OPERANDS
    - USAGE
    - EXIT STATUS
    - SEE ALSO

- A manual page consists of several sections.

Man Page Section Names	
NAME	FILES
SYNOPSIS	VERSIONS
CONFIGURATION	CONFORMING TO
DESCRIPTION	NOTES
OPTIONS	BUGS
EXIT STATUS	EXAMLE
RETURN VALUE	AUTHORS
ERRORS	SEE ALSO
ENVIRONMENT	

man -k

- man **-k**, **--apropos**

Equivalent to **apropos**. Search the short manual page descriptions for keywords and display any matches.

Knowing the user Terminal

Terminal (tty) Command:

- tty - print the file name of the terminal connected to standard input.
- E.g. /dev/pty1

Displaying and Setting terminal characteristics

Command: stty

- stty - change and print terminal line settings/ Print or change terminal characteristics.
- E.g.

  $ stty

  speed 38400 baud; line = 0;

  ixany

- $ stty -a

speed 38400 baud; rows 58; columns 192; line = 0;

intr = ^C; quit = ^\; erase = ^?; kill = ^U; eof = ^D; eol = <undef>; eol2 = <undef>; swtch = ^Z; start = ^Q; stop = ^S; susp = ^Z; rprnt = ^R; werase = ^W; lnext = ^V; flush = ^O;

min = 1; time = 0;

-parenb -parodd cs8 -hupcl -cstopb cread -clocal -crtscts

-ignbrk brkint -ignpar -parmrk -inpck -istrip -inlcr -igncr icrnl ixon -ixoff -iuclc ixany imaxbel

opost -olcuc -ocrnl onlcr -onocr -onlret -ofill -ofdel nl0 cr0 tab0 bs0 vt0 ff0

isig icanon iexten echo echoe echok -echonl -noflsh -tostop echoctl echoke

man -f

- Each manual page has a short description available within it.
- **whatis** searches the manual page names and displays the manual page descriptions of any name matched.
- **whatis** - display one-line manual page descriptions

Unix Shell Programming(USP)

15CS35

What is Unix?

- Unix (trademarked as UNIX) is a family of multitasking, multiuser computer operating systems that derive from the original AT&T Unix, developed starting in the 1970s at the Bell Labs research center by Ken Thompson, Dennis Ritchie, and others.

What is UNIX Shell?

- A **Unix shell** is a command-line interpreter.
- Users direct the operation of the computer by entering commands as **text** for a command line interpreter to execute, or by creating **text scripts** of one or more such commands.
- In some operating systems, the **command interpreter** is **called** the **shell**.

What is a shell script?

- A **shell script** is a computer program designed to be run by the Unix **shell.**
- Typical operations performed by **shell scripts** include file manipulation, program execution, and printing text.

What is Command?

- A **command** is an instruction given by a user telling a **computer** to do something.
- In computers, a **command** is a specific order from a user to the computer's operating system or to an application to perform a service.
- A **command** is a specific instruction given to a computer application to perform some kind of task or function.

What is an Operating System(OS)

- An OS is the software (system) that manages the computer's hardware and provides a convenient and safe environment for running programs.
- It acts as an interface between programs and the hardware resources(Memory, I/O devices, CPU) that these programs access.
- **It acts as an interface between user and the computer system.**

The role of an OS when a program is run

- **Allocate memory** for the program and load the program to the allocated memory.
- **Load CPU registers** with control information's related to the program.
- **Keep track of the last instruction** executed by a program.
- **Provides any other H/W** if requested by the program.
- **Clean up the memory and CPU registers** after the program has completed execution.

The Unix OS

- Is a giant.
- Is way ahead of others (OS) in sheer power.
- Have practically everything that an OS should have and several features which other Operating Systems never had.
- Its richness and elegance go beyond the commands and tools that constitute it.
- Its simplicity permeate the entire system.
- It runs practically on every H/W.
- Provide inspiration to the Open Source Movement.

Contd..

Disadvantage/ Downside

- Makes many demands of the users.
- Even for the experienced computer professional it requires a different type of commitment to understand the subject.
- It introduces concepts not known to the computing community before.
- Uses numerous symbols whose meaning are anything but obvious.
- It achieves unusual tasks with a few keystrokes.

Contd..

- Often, it doesn’t tell you whether you are right or wrong.
- It doesn’t warn you of the consequences of your actions.

What is Computer Data?

- **Computer data** is information processed or stored by a **computer**. This information may be in the form of text documents, images, audio clips, software programs, or other types of **data**. **Computer data** may be processed by the **computer's** CPU and is stored in files and folders on the **computer's** hard disk.

What all can be done using UNIX?

- Computer aided design, manufacturing control systems, laboratory simulations, even the Internet itself, all began life with and because of UNIX systems.
- Today, without UNIX systems, the Internet would come to a screeching halt. Most telephone calls could not be made, electronic commerce would grind to a halt and there would have never been "Jurassic Park"!

What is a Computer File?

- A **file** is an object on a **computer** that stores data, information, settings, or commands used with a **computer** program.
- A **file** is a named collection of related information that is recorded on secondary storage such as magnetic disks, magnetic tapes and optical disks. In general, a **file** is a sequence of bits, bytes, lines or records whose meaning is defined by the **files** creator and user.

List of acronyms and abbreviations

- ***IEEE CS***– *Institute of Electrical and Electronics Engineers Computer Society.*
- ***ISO*** – International Organization for Standardization.
- ***IEC*** – International Electrotechnical Commission.
- ***ICT*** – *Information and Communication Technology*.
- ***IT*** – *Information Technology.*
- ***SUS*** – *Single UNIX Specification.*
- ***POSIX*** – *Portable Operating System Interface.*
- ***XPG*** – *X/Open Portability Guide.*

The Single UNIX Specification

- Focus on application development.

Austin Group

- The **Austin Group** or the **Austin Common Standards Revision Group** is a joint technical working group formed to develop and maintain a common ***revision of POSIX.1 and parts of the Single UNIX Specification***.

- The approach to specification development is "**write once, adopt everywhere**", with the deliverables being a set of specifications that carry both
  - The IEEE POSIX designation,
  - The Open Group's Technical Standard designation, and
  - the ISO/IEC designation.
- The new set of specifications is simultaneously ISO/IEC/IEEE 9945, and forms the core of the ***Single UNIX Specification Version 3***.
- The IEEE formerly designated this standard as 1003.1.

POSIX – a registered trademark of IEEE

- Is an acronym for **P**ortable **O**perating **S**ystem **I**nterface**.**
- Is a family of **standards** specified by the IEEE Computer Society **for** maintaining **compatibility between operating systems**.

POSIX – a registered trademark of IEEE

- **POSIX defines**
  - The **A**pplication **P**rogramming **I**nterface (API), along with
  - Command line shells and utility interfaces,

  for software compatibility with variants of Unix and other operating systems.

- **What is POSIX.1?**
  - A synonym for **IEEE Std 1003.1-1988**, the **BASE STANDARD** upon which **POSIX family of standards** has been built.
  - Is the standard for an Application Program Interface in the C language.
- **What is IEEE POSIX 1003.2 Shell and Utilities?**
  - This standard defines a **standard source level interface** to the shell and utility functionality required by application programs, including shell scripts.

Application Programming Interface-API

- In computer programming, an **API** is a set of subroutine definitions, protocols, and tools for building application software.
- Defines **methods of communication** between various software components.

Application Programming Interface-API

- A good API makes it easier to develop a computer program by providing all the building blocks, which are then put together by the programmer.
- An API may be for a **web-based system, operating system, database system, computer hardware or software library**.

X/Open

- **X/Open Company, Ltd.**, originally the **Open Group for Unix Systems** was a consortium founded by several European UNIX systems manufacturers in 1984 to identify and ***promote open standards*** in the field of information technology.
- More specifically, the original **aim** *was to define a single specification for operating systems derived from UNIX*, to increase the ***interoperability*** of applications and reduce the ***cost*** of porting software.
- Its original members were **B**ull, **I**CL, **S**iemens, **O**livetti, and **N**ixdorf—a group sometimes referred to as **BISON**. Philips and Ericsson joined soon afterwards, at which point the name X/Open was adopted.

POSIX – a registered trademark of IEEE

- Is an acronym for **P**ortable **O**perating **S**ystem **I**nterface**.**
- Is a family of **standards** specified by the IEEE Computer Society **for** maintaining **compatibility between operating systems**.

Note: IEEE – Institute of Electrical and Electronics Engineers

POSIX – a registered trademark of IEEE

- POSIX defines
- The application programming interface (API), along with
- Command line shells and utility interfaces,

for software compatibility with variants of Unix and other operating systems.

- **What is POSIX.1?**
  - A synonym for IEEE Std 1003.1-1988, the **base standard** upon which **POSIX family of standards** has been built.
  - Is the standard for an Application Program Interface in the C language.
- **What is IEEE POSIX 1003.2 Shell and Utilities?**
  - This standard defines a standard source level interface to the shell and utility functionality required by application programs, including shell scripts.

Application Programming Interface-API

- In computer programming, an **API** is a set of subroutine definitions, protocols, and tools for building application software.
- Defines **methods of communication** between various software components.

Application Programming Interface-API

- A good API makes it easier to develop a computer program by providing all the building blocks, which are then put together by the programmer.
- An API may be for a **web-based system, operating system, database system, computer hardware or software library**.

Library (computing)

- Is a collection of non-volatile resources used by computer programs, often to develop software.
- Is also a collection of **implementations of behavior**, written in terms of a language, that has a well-defined interface by which the behavior is invoked.
- For example, in a simple imperative language such as C, the behavior in a library is invoked by using C's normal function-call.

System call

- In computing, a **system call** is the programmatic way in which a computer **program requests a service from the kernel** of the operating system it is executed on.
- This may include
  - hardware-related services (for example, accessing a hard disk drive),
  - creation and execution of new processes, and
  - communication with integral kernel services
    - such as process scheduling.
- System calls provide an essential interface between a process and the operating system.

Introduction - POSIX.1-2008

- **1.1 Scope**

  POSIX.1-2008 defines a standard operating system interface and environment, including a command interpreter (or "shell"), and common utility programs to support applications portability at the source code level. It is intended to be used by both application developers and system implementers.

- POSIX.1-2008 comprises four major components (each in an associated volume):
  - General terms, concepts, and interfaces common to all volumes of POSIX.1-2008, including utility conventions and C-language header definitions, are included in the Base Definitions volume of POSIX.1-2008.
  - Definitions for system service functions and subroutines, language-specific system services for the C programming language, function issues, including portability, error handling, and error recovery, are included in the System Interfaces volume of POSIX.1-2008.
  - Definitions for a standard source code-level interface to command interpretation services (a "shell") and common utility programs for application programs are included in the Shell and Utilities volume of POSIX.1-2008.
  - Extended rationale that did not fit well into the rest of the document structure, containing historical information concerning the contents of POSIX.1-2008 and why features were included or discarded by the standard developers, is included in the Rationale (Informative) volume of POSIX.1-2008.

- POSIX.1-2008 describes the external characteristics and facilities that are of importance to application developers, rather than the internal construction techniques employed to achieve these capabilities. Special emphasis is placed on those functions and facilities that are needed in a wide variety of commercial applications.

Austin Group

- The **Austin Group** or the **Austin Common Standards Revision Group** is a joint technical working group formed to develop and maintain a common ***revision of POSIX.1 and parts of the Single UNIX Specification***.

- The approach to specification development is "**write once, adopt everywhere**", with the deliverables being a set of specifications that carry both
  - The IEEE POSIX designation,
  - The Open Group's Technical Standard designation, and
  - the ISO/IEC designation.
- The new set of specifications is simultaneously ISO/IEC/IEEE 9945, and forms the core of the ***Single UNIX Specification Version 3***.
- The IEEE formerly designated this standard as 1003.1.

POSIX – a registered trademark of IEEE

- Is an acronym for **P**ortable **O**perating **S**ystem **I**nterface**.**
- Is a family of **standards** specified by the IEEE Computer Society **for** maintaining **compatibility between operating systems**.

Note: IEEE – Institute of Electrical and Electronics Engineers

POSIX – a registered trademark of IEEE

- **POSIX defines**
  - The **A**pplication **P**rogramming **I**nterface (API), along with
  - Command line shells and utility interfaces,

  for software compatibility with variants of Unix and other operating systems.

- **What is POSIX.1?**
  - A synonym for **IEEE Std 1003.1-1988**, the **BASE STANDARD** upon which **POSIX family of standards** has been built.
  - Is the standard for an Application Program Interface in the C language.
- **What is IEEE POSIX 1003.2 Shell and Utilities?**
  - This standard defines a **standard source level interface** to the shell and utility functionality required by application programs, including shell scripts.

Application Programming Interface-API

- In computer programming, an **API** is a set of subroutine definitions, protocols, and tools for building application software.
- Defines **methods of communication** between various software components.

Application Programming Interface-API

- A good API makes it easier to develop a computer program by providing all the building blocks, which are then put together by the programmer.
- An API may be for a **web-based system, operating system, database system, computer hardware or software library**.

Library (computing)

- Is a collection of non-volatile resources used by computer programs, often to develop software.
- Is also a collection of **implementations of behavior**, written in terms of a language, that has a well-defined interface by which the behavior is invoked.
- For example, in a simple imperative language such as C, the behavior in a library is invoked by using C's normal function-call.

System call

- In computing, a **system call** is the programmatic way in which a **computer program requests a service from the kernel** of the operating system it is executed on.
- This may include
  - hardware-related services (for example, accessing a hard disk drive),
  - creation and execution of new processes, and
  - communication with integral kernel services
    - such as process scheduling.
- System calls provide an essential ***interface between a process and the operating system***.

- On Unix, Unix-like and other POSIX-compliant operating systems, popular system calls are:
  - **open,**
  - **read,**
  - **write,**
  - **close,**
  - **wait,**
  - **exec,**
  - **fork,**
  - **exit,**
  - **kill.**

X/Open

- **X/Open Company, Ltd.**, originally the **Open Group for Unix Systems** was a consortium founded by several European UNIX systems manufacturers in 1984 to identify and ***promote open standards*** in the field of information technology.
- More specifically, the original **aim** *was to define a single specification for operating systems derived from UNIX*, to increase the ***interoperability*** of applications and reduce the ***cost*** of porting software.
- Its original members were **B**ull, **I**CL, **S**iemens, **O**livetti, and **N**ixdorf—a group sometimes referred to as **BISON**. Philips and Ericsson joined soon afterwards, at which point the name X/Open was adopted.

The Single UNIX Specification

- Focus on application development.

QUIZ

Quiz Questions

1. A _____ is a combination of hardware and software that let users work with a computer.
2. The _____ is a combination of devices that can be seen or touched.
3. Software is divided in to _____ and _____ types.
4. The _____ manages the resources of a computer.

Quiz Questions

5. The three Unix environments are _____, _____, and _____.
6. Unix components are ____ , ____ , ____ and ____ .
7. ____ is the heart of Unix OS.
8. Shell executes ___ and _____ .
9. Three standard shells are ___ , ___ , and ____.

Quiz Questions

10. _____ programs provides support process for the users.
11. _____ programs provide extended capability to the system.
12. ___ have places and ____ have life.
13. There can be multiple kernels and shells running on your system. True or False.
14. _____ and _____ designed Unix OS.
15. Unix OS is implemented using ___ and ___ languages.

Answers

Question No.	Answer	Question No.	Answer
1.	Computer System	9.	Bourne, C and Korn
2.	Hardware	10.	Utilities
3.	System and Application	11.	Application
4.	OS/ Kernel	12.	File and Process
5.	Personal, time-sharing and Client-Server	13.	False
6.	Kernel, shell, Utilities and Applications.	14.	Dennis Ritchie and Ken Thompson
7.	Kernel	15.	C and Assembly
8.	Commands and Scripts		

SHELL PROGRAMMING

MODULE-4

BASH

- NAME

  bash - GNU Bourne-Again SHell

- SYNOPSIS

  bash [options] [command_string | file]

- DESCRIPTION

  Bash is an sh-compatible command language interpreter that executes commands read from the standard input or from a file.

DEFINITIONS

The following definitions are used throughout the rest of this document.

- **blank** - A space or tab.
- **word** - A sequence of characters considered as a single unit by the shell. Also known as a **token**.

name - A word consisting only of alphanumeric characters and underscores, and beginning with an alphabetic character or an underscore. Also referred to as an **identifier**.

- **metacharacter**

A character that, when unquoted, separates words. One of the following:

| & ; ( ) < > space tab

- **control operator**

A token that performs a control function. It is one of the following symbols:

|| & && ; ;; ( ) | |& <newline>

- **RESERVED WORDS**

Reserved words are words that have a special meaning to the shell. The following words are recognized as reserved when unquoted and either the first word of a simple or the third word of a case or for command:

!	**elif**	**function**	**until**	**[**
case	**else**	**if**	**while**	**]**
coproc	**esac**	**in**	**{**	
do	**fi**	**select**	**}**	
done	**for**	**then**	**time**	

- **Pipelines**

A pipeline is a sequence of one or more commands separated by one of the control operators **| or |&.**

- The format for a pipeline is:

[time [-p]] [!] command [[| | |&] command2 ...]

- **Shell Function Definitions**

A shell function is an object that is called like a simple command and executes a compound command with a new set of positional parameters.

- Shell functions are declared as follows:
  - **name () compound-command [redirection]**
  - **function name [()] compound-command [redirection]**

- **QUOTING**
  - **Quoting is used to remove the special meaning of certain characters or words to the shell.**
  - Quoting can be used to disable special treatment for special characters, to prevent reserved words from being recognized as such, and to prevent parameter expansion.
- **There are three quoting mechanisms:**
  - *the escape character (\),*
  - *single quotes (‘’), and*
  - *double quotes (“”).*

- A non-quoted backslash (\) is the escape character. It preserves the literal value of the next character that follows, with the exception of <newline>. If a \<newline> pair appears, and the backslash is not itself quoted, the \<newline> is treated as a line continuation (that is, it is removed from the input stream and effectively ignored).

- Enclosing characters in single quotes preserves the literal value of each character within the quotes. A single quote may not occur between single quotes, even when preceded by a backslash.

- Enclosing characters in double quotes preserves the literal value of all characters within the quotes, with the exception of $, `, \, and, when history expansion is enabled, !.
- The characters $ and ` retain their special meaning within double quotes.
- The backslash retains its special meaning only when followed by one of the following characters: $, `, ", \, or <newline>.

- A double quote may be quoted within double quotes by preceding it with a backslash. If enabled, history expansion will be performed unless an ! appearing in double quotes is escaped using a backslash. The backslash preceding the ! is not removed.
- The special parameters * and @ have special meaning when in double quotes.

- **PARAMETERS**

A parameter is an entity that stores values. It can be a ***name***, a ***number***, or one of the ***special characters*** under Special Parameters.

- **A variable is a parameter denoted by a name**.
  - A variable has a value and zero or more attributes. Attributes are assigned using the declare builtin command.

- **A parameter is set if it has been assigned a value**. **The null string is a valid value**. Once a variable is set, it may be unset only by using the **unset builtin command**.
- A variable may be assigned to by a statement of the form: **name=[value]**
- **If value is not given, the variable is assigned the null string**.

- **Positional Parameters**
  - **A positional parameter is a parameter denoted by one or more digits, other than the single digit 0. ($1,$2,…$9,${10},${11},….).**
  - **Positional parameters are assigned from the shell's arguments when it is invoked**, and may be reassigned using the set builtin command.
  - Positional parameters may not be assigned to with assignment statements. The positional parameters are temporarily replaced when a shell function is executed.

- **Special Parameters:** The shell treats several parameters specially. These parameters may only be referenced; assignment to them is not allowed.

Parameter	Significance
$*	Expands to the positional parameters, starting from one. **Can be used with or without double quotes. Note: default 9 positional parameters viz. $1…..$9. For 2 digits PP syntax: ${10}, ${11}, and so on.**
$@	Expands to the positional parameters, starting from one. **Can be used with or without double quotes. Note: default 9 positional parameters viz. $1…..$9**
$#	Expands to the number of positional parameters in decimal.
$?	Expands to the exit status of the most recently executed foreground pipeline.
$$	Expands to the process ID of the shell.
$!	Expands to the process ID of the job most recently placed into the background
$0 (Zero)	Expands to the name of the shell or shell script.

Variables

- Variable is a memory location where values can be stored.

Classification:

- User-defined - E.gs. Hjr,_hjr,hjr123,_hjr123.
- Predefined
  - Shell variables, to configure the shell.
  - Environmental variables to configure the shell environment.

- Expressions – are a sequence of operands and operators that reduces to a single value.
- Operators
  - Mathematical
  - Relational
  - File test
  - Logical

Mathematical Operators – to compute a numeric value.

Operator	Description
+	Value is sum of two numbers.
-	Value is difference of two numbers.
*	Value is product of two numbers.
/	Value is quotient of two numbers.
%	Value is remainder of two numbers.

Relational Operators- to compare two values and return either true or false.

Operator	Meaning	String Interpretation
>	Greater than	
>=	Greater than or equal	
<	Less than	
<=	Less than or equal	
==	equal	=
!=	Not equal	!=
	String length not zero	-n
	String length zero	-z

Single Unix Specification(SUS)

- SUS is the collective name of a family of standards for computer OS, compliance with which is required to qualify for using UNIX trademark.
- The **core specifications** of the SUS are developed and maintained by the Austin Group, which is a joint working group of IEEE, ISO JTC 1 SC22 and The Open Group.

Motivation

- To standardize **operating system interfaces** for software designed for variants of the Unix operating system.

Need

- In Enterprises the requirements were:
  - **Interoperability of applications, programs.**
  - **Reduce the cost of porting S/W.**
- Enterprises using computers wanted to be able to develop programs that could be used on the computer systems of different manufacturers without re-implementing the programs.

Why Unix

- Unix was selected as the basis for a standard system interface partly because it was manufacturer-neutral.

Two Standards- XPG and POSIX

for maintaining compatibility between operating systems.

XPG – X/Open Portability Guide	POSIX- Portable Operating System Interface
Created by X/Open, the open group of consortium of vendors and users in Europe.	Standards specified by IEEE Computer Society (IEEE-Institute of Electrical and Electronics Engineers)
List of Companies- BISON (Bull, ICL, Siemens, Olivetti and Nixdorf)	Two most cited standards: POSIX.1 and POSIX.2 for software compatibility with variants of Unix and other operating systems.
Products Conforming to this specifications were branded as UNIX95, UNIX97 and so on.	POSIX.1 defines the application programming interface (API). POSIX.2 defines command line shells and utility interfaces.

UNIX AND SHELL PROGRAMMING

Subject Code 15CS35

CREDITS – 04

Course objectives:

This course will enable students to

- Understand the UNIX Architecture, File systems and use of basic Commands.
- Use of editors and Networking commands.
- Understand Shell Programming and to write shell scripts.
- Understand and analyze UNIX System calls, Process Creation, Control & Relationship.

Module -1
Teaching Hours - 10Hours

- Introduction,

- Brief history.

Unix Components/Architecture

- Unix is made up of three components
  - The kernel
  - The shell and
  - Programs.

- Features of Unix.

- The UNIX
- Environment and UNIX Structure,

- Posix and Single Unix specification.

- The login prompt.

- General features of Unix commands/ command structure.

- Command arguments and
- options.

- Understanding of some basic commands such as echo, printf, ls, who, date,
- passwd, cal, Combining commands.

- Meaning of Internal and external commands.

- The type
- command: knowing the type of a command and locating it.

- The man command knowing
- more about Unix commands and using Unix online manual pages.

- The man with keyword
- option and whatis.

- The more command and using it with other commands.

- Knowing the
- user terminal, displaying its characteristics and setting characteristics.

- Managing the nonuniform
- behaviour of terminals and keyboards.

- The root login. Becoming the super user: su
- command.

- The /etc/passwd and /etc/shadow files.

- Commands to add, modify and delete
- users.

Module -2

- Unix files.

- Naming files.

- Basic file types/categories.

- Organization of files.

- Hidden files.

- Standard directories.

- Parent child relationship.

- The home directory and the HOME
- variable.

- Reaching required files- the PATH variable,

- manipulating the PATH, permissions.

- Relative
- and absolute pathnames.

- Directory commands – pwd, cd, mkdir, rmdir commands.

- The dot
- (.) and double dots (..) notations to represent present and parent directories and their usage
- in relative path names.

- File related commands – cat, mv, rm, cp, wc and od commands.

- File attributes and permissions and knowing them.

- The ls command with options.

- Changing file permissions: the relative and absolute permissions changing methods.

- Recursively changing file permissions.

- Directory Permission

Module -2

- Unix files.

- Naming files.

- Basic file types/categories.

- Organization of files.

- Hidden files.

- Standard directories.

- Parent child relationship.

- The home directory and the HOME
- variable.

- Reaching required files- the PATH variable,

- manipulating the PATH, permissions.

- Relative
- and absolute pathnames.

- Directory commands – pwd, cd, mkdir, rmdir commands.

- The dot
- (.) and double dots (..) notations to represent present and parent directories and their usage
- in relative path names.

- File related commands – cat, mv, rm, cp, wc and od commands.

- File attributes and permissions and knowing them.

- The ls command with options.

- Changing file permissions: the relative and absolute permissions changing methods.

- Recursively changing file permissions.

- Directory Permission

Module – 3

- The vi editor. Basics.

- The .exrc file.

- Different ways of invoking and quitting vi.

- Different
- modes of vi. Input mode commands. Command mode commands. The ex mode commands.

- Illustrative examples Navigation commands.

- Repeat command.

- Pattern searching.

- The
- search and replace command.

- The set, map and abbr commands

- . Simple examples using
- these commands.

- The shells interpretive cycle.

- Wild cards and file name generation.

- Removing the special
- meanings of wild cards.

- Three standard files and redirection. Connecting commands: Pipe.

- Splitting the output: tee.

- Command substitution. Basic and Extended regular expressions.

- The grep, egrep.

- Typical examples involving different regular expressions.

Module-4

- Shell programming.

- Ordinary and environment variables.

- The .profile.

- Read and readonly
- commands.

- Command line arguments.

- exit and exit status of a command

- . Logical operators

- for conditional execution.

- The test command and its shortcut.

- The if, while, for and case
- control statements.

- The set and shift commands and handling positional parameters.

- The
- here (<<) document and trap command.

- Simple shell program examples

- . File inodes and
- the inode structure.

- File links – hard and soft links.

- Filters. Head and tail commands.

- Cut
- and paste commands.

- The sort command and its usage with different options.

- The umask
- and default file permissions.

- Two special files /dev/null and /dev/tty.

UNIX AND SHELL PROGRAMMING

Subject Code 15CS35

CREDITS – 04

Course objectives:

This course will enable students to

- Understand the UNIX Architecture, File systems and use of basic Commands.
- Use of editors and Networking commands.
- Understand Shell Programming and to write shell scripts.
- Understand and analyze UNIX System calls, Process Creation, Control & Relationship.

Module -1
Teaching Hours - 10Hours

- Introduction,

- Brief history.

- Unix Components/Architecture.

- Features of Unix.

- The UNIX
- Environment and UNIX Structure,

- Posix and Single Unix specification.

- The login prompt.

- General features of Unix commands/ command structure.

- Command arguments and
- options.

- Understanding of some basic commands such as echo, printf, ls, who, date,
- passwd, cal, Combining commands.

- Meaning of Internal and external commands.

- The type
- command: knowing the type of a command and locating it.

- The man command knowing
- more about Unix commands and using Unix online manual pages.

- The man with keyword
- option and whatis.

- The more command and using it with other commands.

- Knowing the
- user terminal, displaying its characteristics and setting characteristics.

- Managing the nonuniform
- behaviour of terminals and keyboards.

- The root login. Becoming the super user: su
- command.

- The /etc/passwd and /etc/shadow files.

- Commands to add, modify and delete
- users.

UNIX **vi** Editor

(**vi**sual editor)

Editor(s)

- Editor is a utility that facilitates the editing task – creation and modifications of text files quickly and efficiently.
- Types of editors
  - Screen editors, e.g. vi, vim, pico, joe, emacs ,ed.
    - Provides a whole screen of text at a time.
  - Line editors, e.g. sed, ex.
    - Are useful to make global changes over a(line or) group of lines.

vi - screen editor

- Is a full screen editor and has **three modes** of operation:
- ***Command mode***
- ***Insert mode / Text mode/ Input Mode***
- ***Last line mode/ ex mode***

Note: the first two are basic modes.

- ***Command mode*** - commands which cause immediate action or command execution over a texts of a open file.
- Actions like save, quit, cursor movements , cut, copy, paste, search, replace, substitution and many others.
- Two aspect of commands:
  - **Commands are not echoed** *on the screen* **except colon(:), Forward slash(/) and question mark(?).**
  - **Most commands should not be followed by Enter key *except :, /, and ?.***

- ***Insert mode / Text mode / Input Mode*** - in which entered text is inserted into the file.
- ***Last line /ex Mode mode*** - The cursor will jump to the last line of the screen and wait for a command.

Command Category

Entering and Replacing text	Undoing
Saving text and quitting	Repeating
Navigation	Pattern searching
Editing Text	Substitution – Search and Replace

Input Mode Commands

Command	Function/ Action
i	Inserts text to left of cursor.
a	Appends text to right of cursor.
I	Inserts text at beginning of line.
A	Appends text at end of line.
o	Open line below.
O	Open line above.
r*ch*	Replace single character under cursor with ch. (no ESC required).
R	Replace text from cursor to right. (Existing text overwritten).
s	Replace single character under cursor with a string.
S	Replace entire line.

ex Mode commands (shown are file save and exit commands only)

Command	Action
:w	Write to file and remain in vi editor.
:x	Write to file and quit vi editor.
:wq	Write to file and quit vi editor.
:q	Quit vi editor when no changes are made to file.
:q!	Quit vi editor abandon all changes to a file.
:sh	Create a shell within the vi editor to execute shell commands. To return back to vi editor **type exit or ctrl-d**.
:!cmd	Execute shell command without leaving vi editor.
:recover	Recover file from crash.
:n1,n2w abc.txt	Write lines n1 to n2 to file abc.txt.
:.w abc.txt	Write current line to file abc.txt.
:$w abc.txt	Write last line to file abc.txt.

Command mode commands

Command Category	
Navigation	
Title/ Name	**Commands**
Cursor Movements	**k, j** (up, down) **h, l** (left, right)
Word Navigation	**b** – move back to beginning of word. **e** - move forward to end of word. **w**- move forward to beginning of word.
Moving to Line Extremes	**0(ZERO)** – move to beginning of the current line. **$**- move to the end of the current line. **|** - position the cursor on a particular column. E.g. 30**|**

Command mode commands

Command Category	
Navigation	
Title/ Name	**Commands**
Scrolling	**Ctrl-f**← scrolls forward. **Ctrl-b** ← scroll backward. **Ctrl-d** ← scroll half page forward. **Ctrl-u** ←scroll half page backward.
Absolute Movements	**G, nG.**
Note: ex mode equivalents of absolute movements	**:number, :$**

Command mode commands

Command Category	
Editing Text	
Title/ Name	**Commands**
Deleting Text	**x-** delete character under cursor. **dd-** delete entire line.
Moving Text (from buffer)	**p-** put the text after cursor. **P**-put the text before cursor.
Copying Text	**yy** – Yank current line
Joining Lines	**J**- removes newline character between the two lines to pull up the line below it.

Command mode commands

Command Category	
Undoing Last Editing	
Title/ Name	**Commands**
Undoing Last Editing	**u**- undo last change. **U**- undo all changes on line.

Command mode commands

Command Category	
Repeating the last command	
Title/ Name	**Commands**
Repeating the last command	• (read dot) – can be applied to repeat the last editing instruction like insertion, deletion, or other actions that modifies the buffer.

Command mode commands

Command Category	
Searching for a pattern	
Title/ Name	**Commands**
Searching for a pattern	**/pat** – search forward for pattern pat.
	?pat – search backward for pattern pat.
	n- Repeat search in same direction of original search.
	N- Repeat search in direction opposite to previous search made.

TRIO of search

- To be carried out at a number of places in combination of
  - / (search command)
  - n (repeat search command)
  - . (repeat last editing command)

Substitution-Search and Replace (:s)

- The general form:

<:><address><s/source_pattern/target_pattern/flags>

Flag – g (global)

Address: .,$,n1..n2

Vi Editor Cheat Sheet

Movement Commands

Character

h, **j**, **k**, **l**	Left, down, up, right

Text

w, **W**, **b**, **B**	Forward, backward by word
e, **E**	End of word
(, **)**	Beginning of next, previous sentence
{, **}**	Beginning of next, previous paragraph
[[, **]]**	Beginning of next, previous section

Lines

0, **$**	First, last position of current line
^	First non-blank character of current line
+, **-**	First character of next, previous line
H	Top line of screen
M	Middle line of screen
L	Last line of screen
*n***H**, *n***L**	Line *n* from top, bottom of screen

Scrolling

[Ctrl]**F**, [Ctrl]**B**	Scroll forward, backward one screen
[Ctrl]**D**, [Ctrl]**U**	Scroll down, up one-half screen
[Ctrl]**E**, [Ctrl]**Y**	Show one more line at bottom, top of window
z[Enter]	Scroll until line with cursor is at top of screen
z.	Scroll until line with cursor is at middle of screen
z-	Scroll until line with cursor is at bottom of screen

Searches

/*pattern*	Search forward for *pattern*
?*pattern*	Search backward for *pattern*
n, **N**	Repeat last search in same, opposite direction
/, **?**	Repeat previous search forward, backward
f*x*	search forward for character *x* in current line
F*x*	search backward for character *x* in current line
t*x*	search forward for character before *x* in current line
T*x*	search backward for character after *x* in current line
;	Repeat previous current-line search
,	Repeat previous current-line search in opposite direction

Line Number

[Ctrl]**G**	Display current line number
*n***G**	Move to line number *n*
G	Move to last line in file
:*n*	move to line number *n*

Marking Position

m*x*	Mark current position as *x*
`*x*	Move cursor to *x*
``	Return to previous mark or context
'*x*	Move to beginning of line containing mark *x*
''	Return to beggining of line containing previous mark

Editing Commands

Insert

i, **a**	Insert text before, after cursor
I, **A**	Insert text at beginning, end of line
o, **O**	Open new line for text below, above cursor

Change

r	Replace with next typed characer

~	Change between uppercase and lowercase
c*m*	Change text block defined by movement command *m* (e.g., cw changes next word)
cc	Change current line
C	Change to end of line
R	Type over characters
s	Delete character and continue typing
S	Delete current line and continue typing

Delete, Move

x	Delete character
X	Delete character to the left of the cursor
d*m*	Delete text block defined by movement command *m* (e.g., dw deletes next word)
dd	Delete current line
D	Delete to end of line
p, **P**	Put deleted text before, after cursor
"*n***p**	Put text from delete buffer number *n* after cursor (for last nine deletions)

Yank (copy)

y*m*	Yank (copy) text block defined by movement command *m* (e.g., yw yanks next word)
yy, **Y**	Yank current line
"*a***yy**	Yank current line into named buffer *a*
p, **P**	Put yanked text before, after cursor
"*a***P**	Put text from buffer *a* before cursor

Other Commands

.	Repeat last edit command
u	Undo last edit
U	Undo changes to current line
J	Join two lines
[Ctrl]**L**, [Ctrl]**R**	Redraw screen

Invoking vi

vi *file*	Invoke vi editor on *file*
vi *file1 file2*	Invoke vi editor on files sequentially
view *file*	Invoke vi editor on *file* in read-only mode
vi -R *file*	Invoke vi editor on *file* in read-only mode
vi -r *file*	Recover *file* and recent edits after system crash
vi + *file*	Open *file* at last line
vi +*n file*	Open *file* at line number *n*
vi +/*pattern file*	Open *file* at *pattern*

Exit and Save Commands

ZZ	Save file and quit
:x	Save file and quit
:wq	Save ("write") file and quit
:w	Save file
:w!	Save file (overriding protection)
:30,60w *newfile*	Save lines 30 through 60 as file *newfile*
:30,60w>> *file*	Append lines 30 through 60 to file *file*
:w %.*new*	Save current buffer named *file* as *file.new*
:q	Quit
:q!	Quit, discarding any changes
Q	Quit vi and invoke ex
:e *file2*	Edit *file2* without leaving vi
:e! *file2*	Discard changes to current file, then edit *file2* without leaving vi
:n	Edit next file
:e!	Discard all changes since last save
:e#	Edit alternate file