MIT-WPU

॥ विश्वशान्तिर्ध्रुवं ध्रुवा ॥

Dr. Vishwanath Karad

MIT WORLD PEACE UNIVERSITY | PUNE

TECHNOLOGY, RESEARCH, SOCIAL INNOVATION & PARTNERSHIPS

Unix Operating Systems

School of Computer Engineering and technology

MIT-WPU
।। विश्वशान्तिर्ध्रुवं ध्रुवा ।।

Unit 5- Unix Operating Systems

- Introduction to Unix Operating System.
- The Unix File System and Process Management.
- Comparison between Windows OS, Unix and Linux.
-
- Basics of shell scripting.

Reference book

The Design of the UNIX Operating System
by Maurice J. Bach

What is UNIX?

- An Operating System (OS)
- Mostly coded in C
- Machine independence
- It provides a number of facilities:
  - management of hardware resources
  - directory and file system
  - loading / execution / suspension of programs

History (Brief)

- **Unix** is a multitasking, multi-user computer operating system
- System originally developed in 1969 by AT & T employees at Bell Labs….
- Key contributers :- Ken Thompson,Dennis Ritchie,Brian Kernighan

Why Use UNIX?

- multi-tasking / multi-user
- lots of software
- networking capability
- graphical (with command line)
- easy to program
- portable (PCs, mainframes, super-computers)

continued

Flavours/examples of Linux

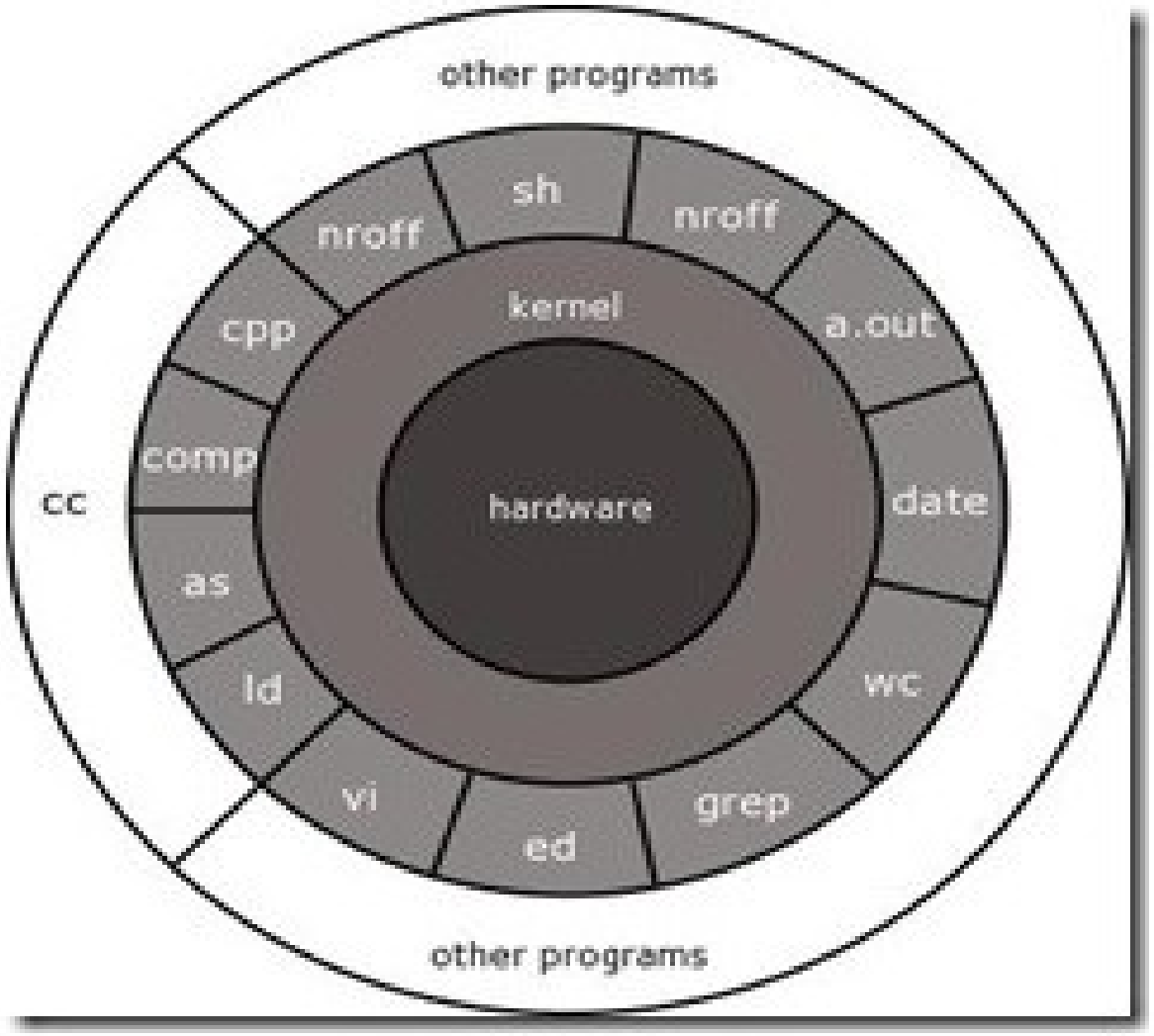
Some Flavours of Unix ie Unix like systems

- AIX by IBM
- BSD/OS
- HP-UX
- Solaris

Architecture of Unix system

Kernel – The kernel interacts with the hardware and performs the tasks like memory management, task scheduling and file management. It is the heart of the OS.

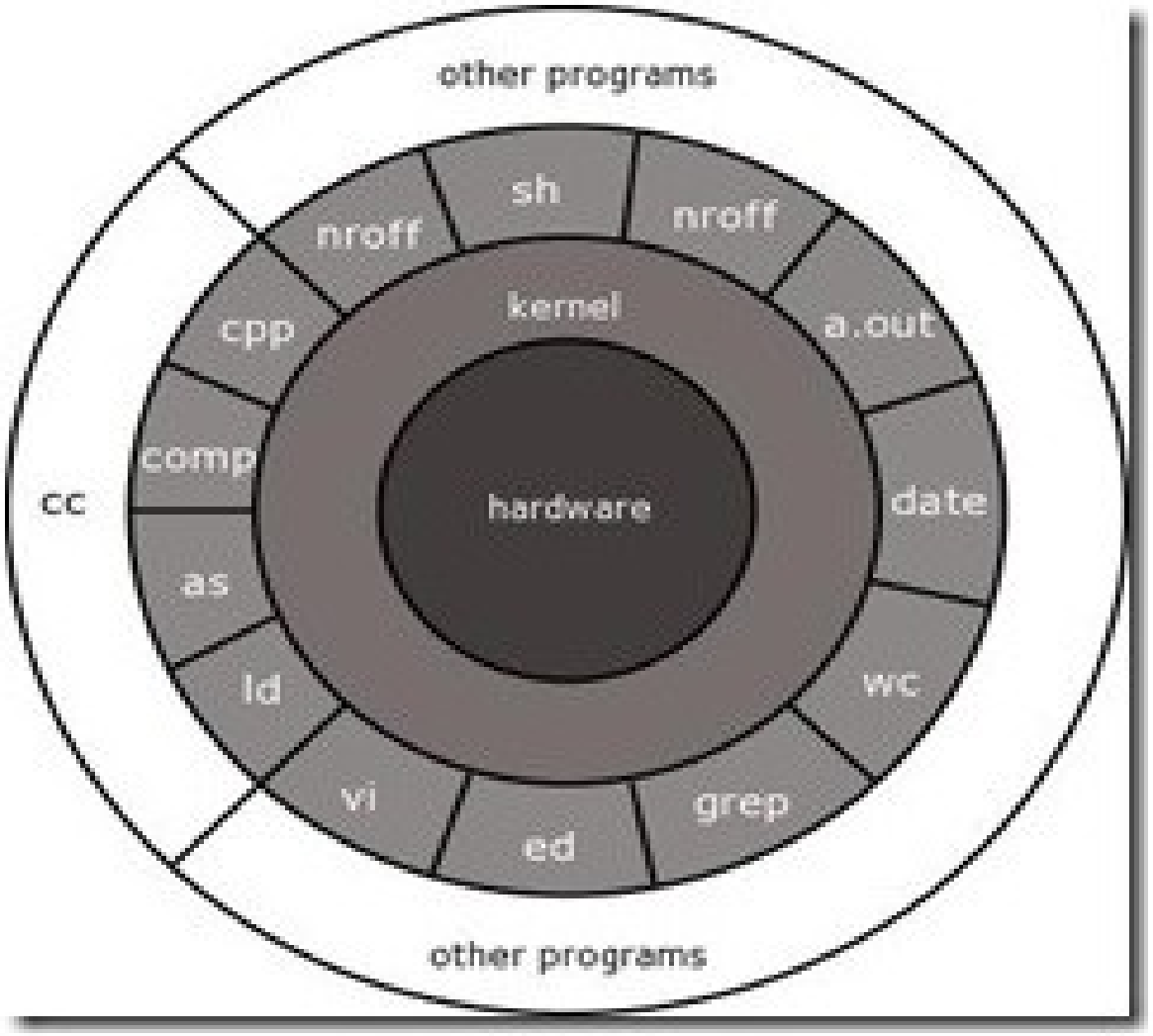
Shell – The shell is the utility that processes your requests. It interprets the command and calls the program that you want.



Continued..

Commands and Utilities − There are various (250 standard) commands and utilities like **cp**, **mv**, **cat** and **grep**, etc.
Programs like shell and editors (ed and vi) interact with kernel by system calls(around 64).
Other application programs like CC can build on top of lower level programs.

Files and Directories − All the data of Unix is organized into files. All files are then organized into directories. These directories are further organized into a tree-like structure called the **filesystem**.



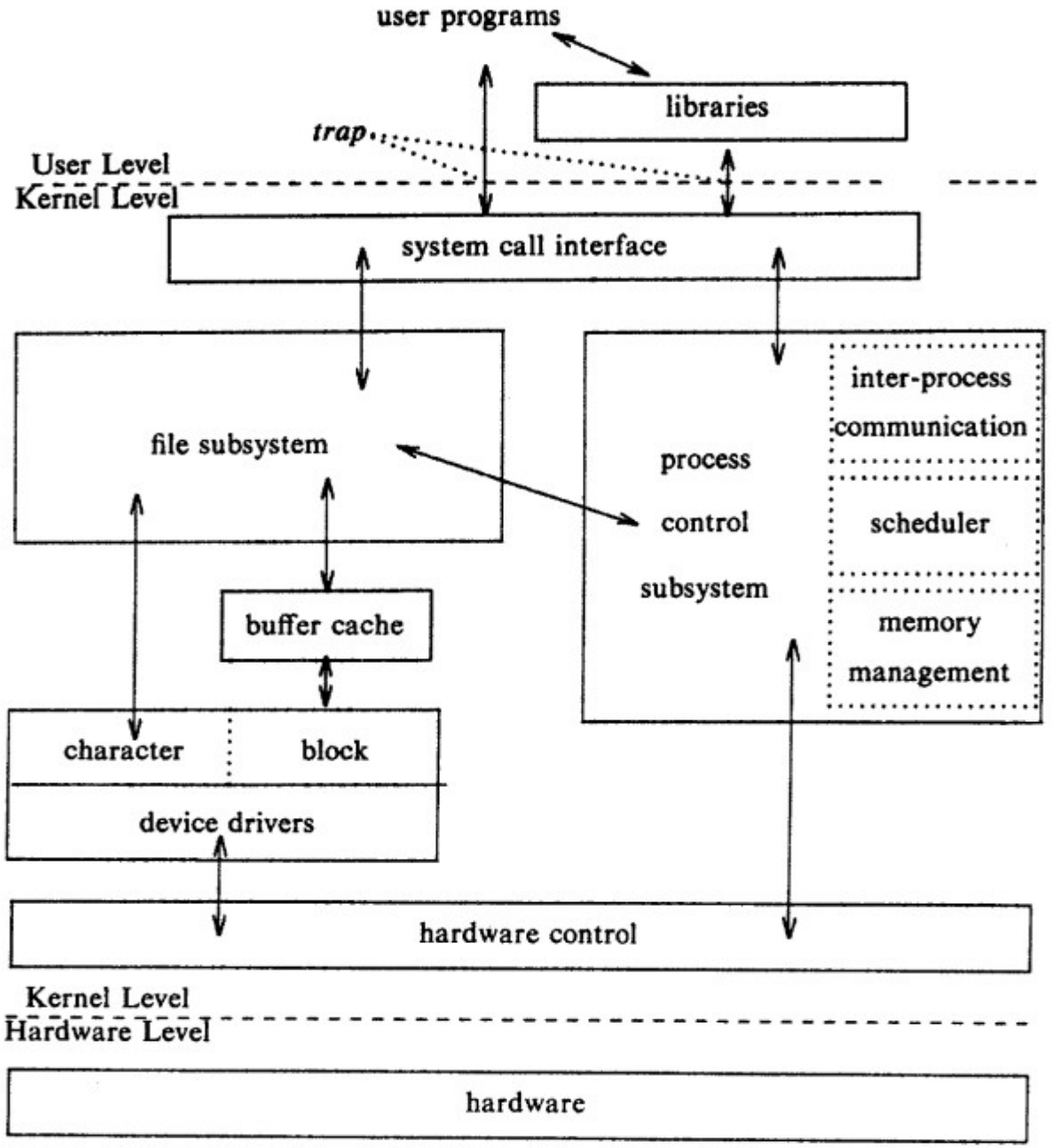
Block diagram of System Kernel

The two entities, **files and processes**, are the two central concepts in the UNIX system model.

There are three levels: **user, kernel, and hardware**.
The system call and library interface represent the border between user programs and the kernel

Libraries map the system calls to the primitives needed to enter OS

The file subsystem **manages files**, allocating file space, administering free space, controlling access to files, and retrieving data for users.

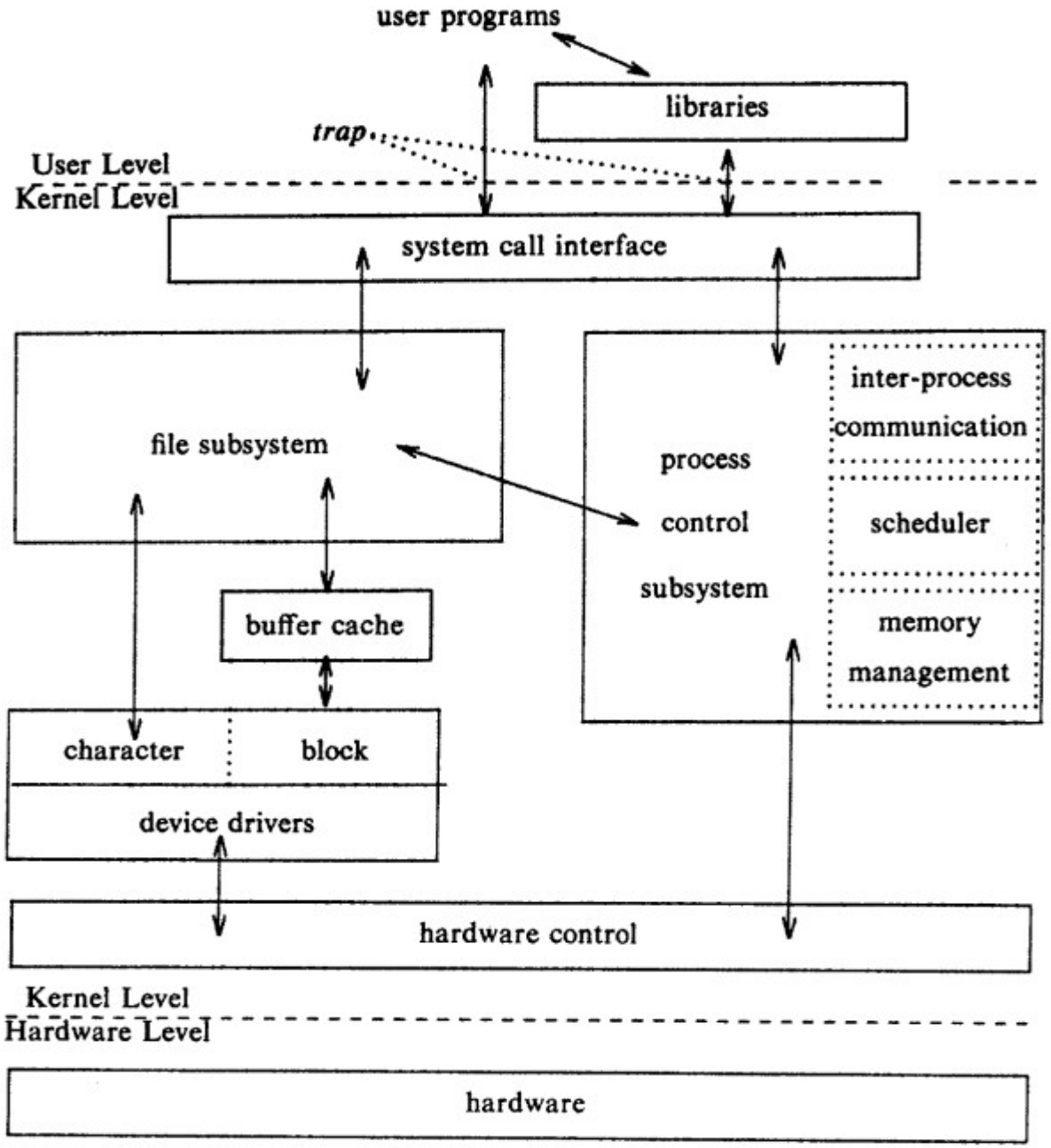


Continued..

Processes interact with the **file subsystem via a specific set of system calls, such as open, close, read, write, stat** (query the attributes of a file), chown (change the record of who owns the file), and chmod (change the access permissions of a file).

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

The hardware control is responsible for **handling interrupts** and for communicating with the machine.



UNIX file system

The UNIX file system is characterized by:

- A hierarchical structure,
- Consistent treatment of file data,
- The ability to create and delete files,
- Dynamic growth of files,
- The protection of file data,
- The treatment of peripheral devices (such as terminals and tape units) as files.

The File

- Ordinary(Regular) Files
- Directory Files
- Device Files

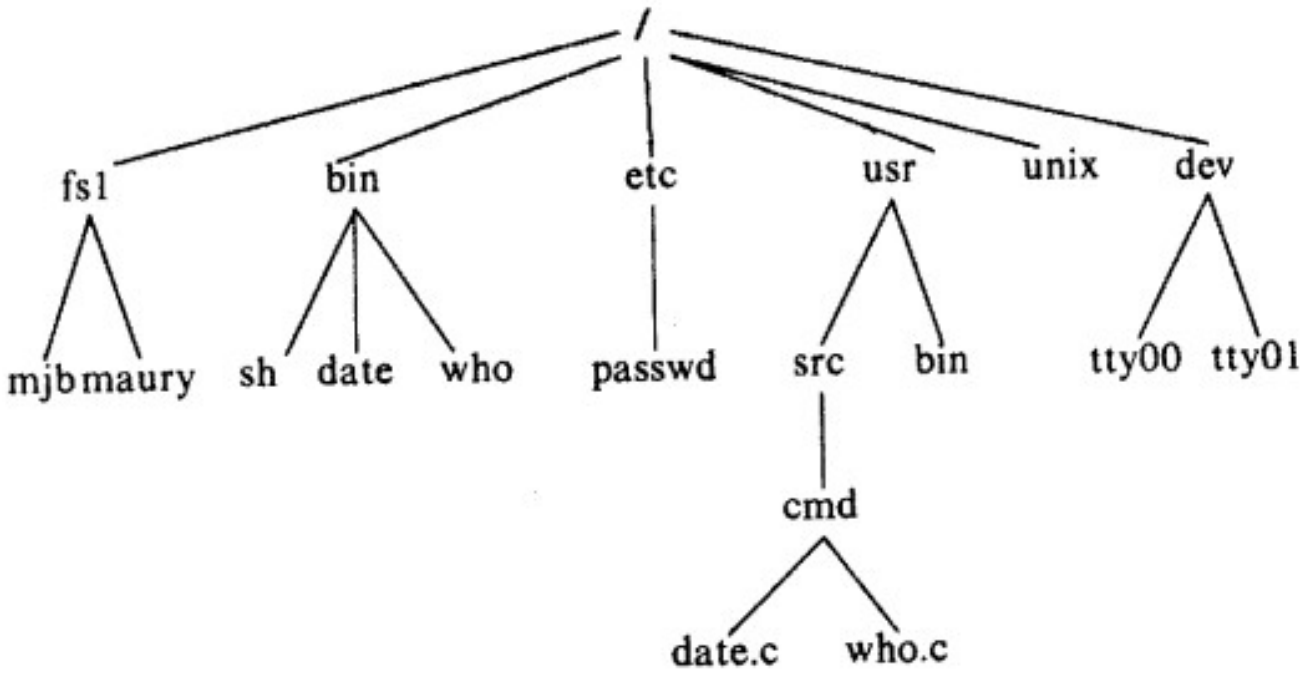
The file system is organized as a tree with a **single root node called “/”;**

Every non-leaf node of the file system structure is directory of files, and files at the leaf nodes of the tree are either directories, files, or special device files.

The name of a file is given by the path name file in the file.

MIT-WPU
।। विश्वशान्तिर्ध्रुवं ध्रुवा ।।

The Parent Child Relationship: simplified UNIX directory/file system



Some System Directories

- `/` *root* directory
- `/bin` commands
- `/etc` system data files (e.g. `/etc/passwd`)
- `/dev` files representing I/O devices

Pathnames

- A ***pathname*** is a sequence of directory names (separated by /'s) which identifies the location of a directory.

- There are two sorts of pathnames
  - A **full path** or **absolute path** is a path that points to the same location on one file system regardless of the working directory or combined paths. **It is usually written in reference to a root directory**.

  - A **relative path** is a path relative to the working directory of the user or application, so the full absolute path will not have to be given.

Absolute Pathnames

Absolute Pathnames:The sequence of directory names between the top of the tree (the *root*) and the directory of interest.

- For example:

  /bin

  /etc/terminfo

  /export/user/home/ad

Relative Pathnames :The sequence of directory names **below** the directory where you are now to the directory of interest.
For example:

If you are interested in the directory proj1:

proj1	if you are in s3910120
s3910120/proj1	if you are in home
home/s3910120/proj1	if you are in user

Access permissions

- Permission to access a file is controlled by access perrnissions associated with the file.
- Access permissions can be set independently to control read, write, an execute

- permission for three classes of users: **the file owner, a file group, an everyone else.**
- Users may create files if directory access permissions allow it.
- The newly created files are leaf nodes of the file system directory structure.

ls –l /etc/passwd

-rw-r--r-- 1 root root 2365 Jul 28 16:19 /etc/passwd

read, write, execute (r w x)

-	rw-	r--	r--
directory	owner	group	everyone

chmod

Standard Files

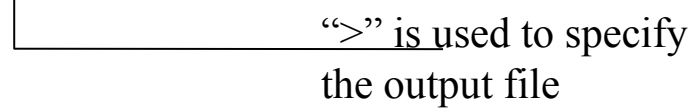
UNIX concept of “standard files”

- standard input (where a command gets its input) –
  standard output (where a command writes it output) -
- standard error (where a command writes error messages)

Redirecting Output

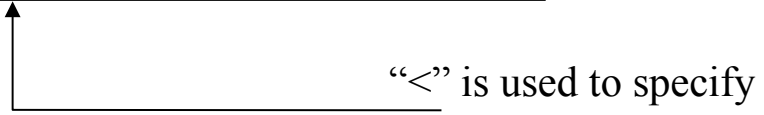
- The output of a command may be sent (piped) to a file:

```
ls -l >output
```



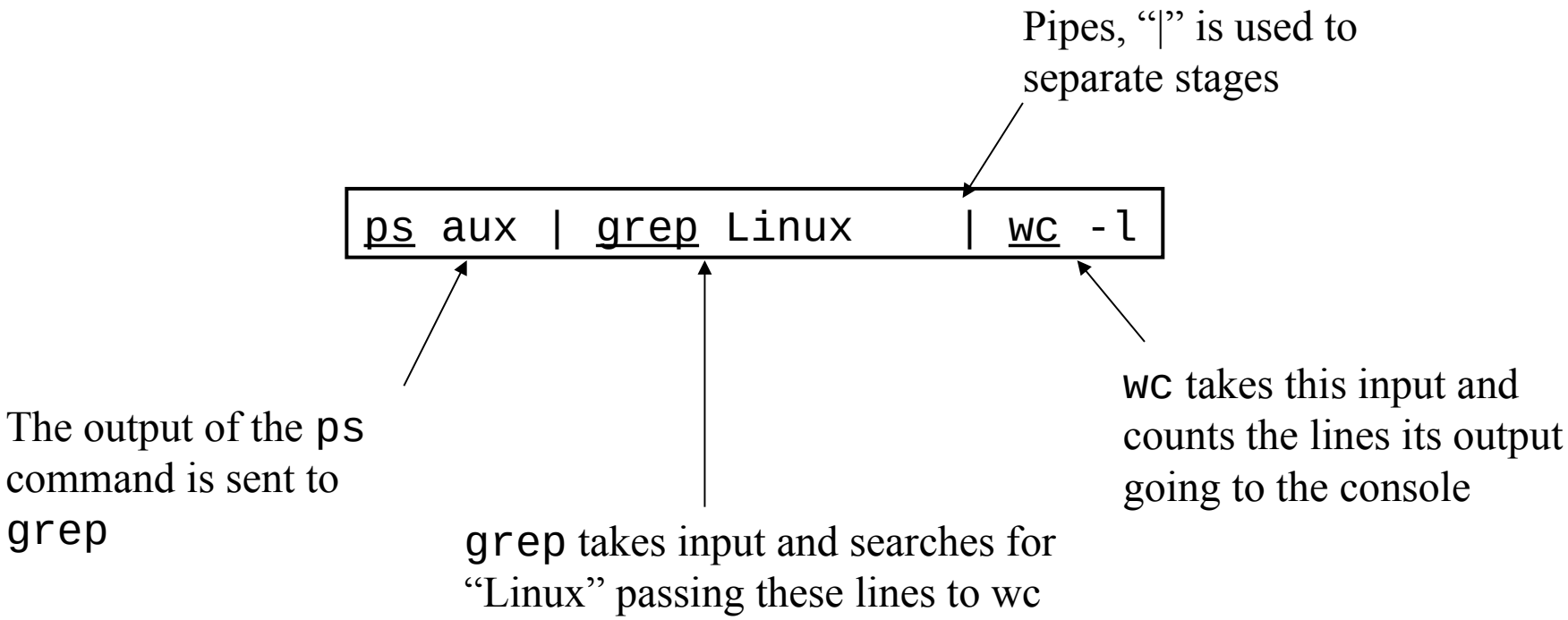
- The input of a command may come (be piped) from a file:

```
wc <input
```



Connecting commands: Pipes

- The output of one command can become the input of another:



Commands and Pathnames

- Commands often use pathnames.

- For example:

```
cat /etc/passwd
```

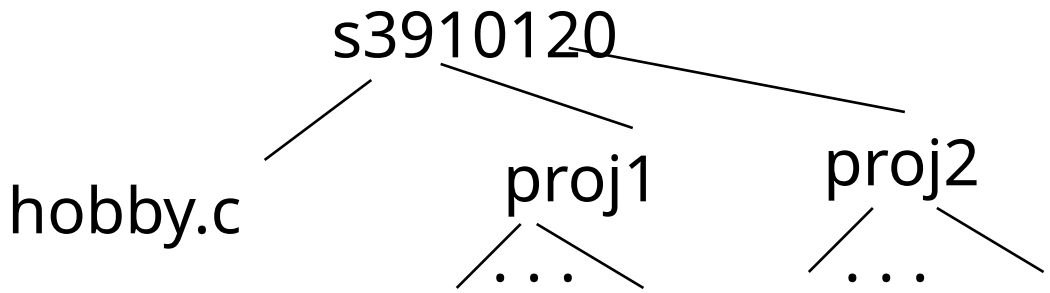
List the password file

Moving between Directories

S3910120 is home directory:

- If you are in directory **s3910120** how do you move to directory **proj1**?

```
cd proj1
```



- You are now in **proj1:** This is called the current working directory.
- **Pwd** Print name of current working directory

- Move back to directory **s3910120** (the parent directory):

```
cd ..
```

- When in **proj1**, move to **proj2** with one command:

```
cd ../proj2
```

../proj2 is a *relative* pathname

Special Directory Names

- `/` The root directory
- `.` The current working directory
- `..` The parent directory (of your current directory)

Examples:

- `cd /` Change to root directory
- `cd ~` Change to home directory
- `cd` (Special case; means `cd ~`)
- `cd ../..` Go up two levels.

Investigate the System

- **Use** `cd`
- **`cat file`** List `file`

```
cd /etc
cat passwd
```

- **`ls`** Directory listing

```
ls                    List current dir.
ls /etc               List /etc
```

Making / Deleting / Renaming Directories

- Usually, you can only create directories (or delete or rename them) in your home directory or directories below it.

`mkdir`	Make a directory
`rmdir`	Delete a directory
`mv`	Rename a directory

Commands to work with files

- **`cat > filename`**
- **`less`**
- **`head`**
- **`tail`**
- **`cp`**
- **`mv`**
- **`rm`**
- **`wc`**
- **`grep`**

Information on Others

- `users` Who else is logged on?
- `who` Information on current users
- `ps` What are people doing?

Mounting

Mounting a device makes it accessible by the computer. This is a software process that enables the operating system to read and write data to the device.

Unmounting a device is the opposite of mounting a device. It takes a mounted device and makes it inaccessible by the computer.

The file named **/etc/fstab** is used to define mountable file systems and devices on startup.

File System

- A file system is consists of a sequence of logical blocks (512/1024 byte etc.)
- A file system has the following structure:

Boot Block	Super Block	Inode List	Data Blocks

File System:

Boot Block

- The beginning of the file system
- Contains bootstrap code to load the operating system
- Initialize the operating system
- Typically occupies the first sector of the disk

Super Block

- Describes the state of a file system
- Describes the size of the file system
- How many files it can store
- Where to find free space on the file system
- Other information

File System:

Inode List

- Inodes are used to access disk files.
- Inodes maps the disk files
- For each file there is an inode entry in the inode list block
- Inode list also keeps track of directory structure

Data Block

- Starts at the end of the inode list
- Contains disk files
- An allocated data block can belong to one and only one file in the file system

Processes

- A process is the execution of a program
- A process is consists of text (machine code), data and stack
- Many process can run simultaneously as kernel schedules them for execution
- Several processes may be instances of one program
- A process reads and writes its data and stack sections, but it cannot read or write the data and stack of other processes
- A process communicates with other processes and the rest of the world via system calls

Process Context

- The context of a process is defined by code,data,stack and its execution environment.
- Context Switch
  - When the kernel decides that it should execute another process, it does a context switch, so that the system executes in the context of the other process
  - When doing a context switch, the kernel saves enough information so that it can later switch back to the first process and resume its execution.

Mode of Process Execution

- The UNIX process runs in two modes:
  - **User mode**
    - Can access its own instructions and data, but not kernel instruction and data
  - **Kernel mode**
    - Can access kernel and user instructions and data
- When a process executes a system call, the execution mode of the process changes from user mode to kernel mode

Mode of Process Execution

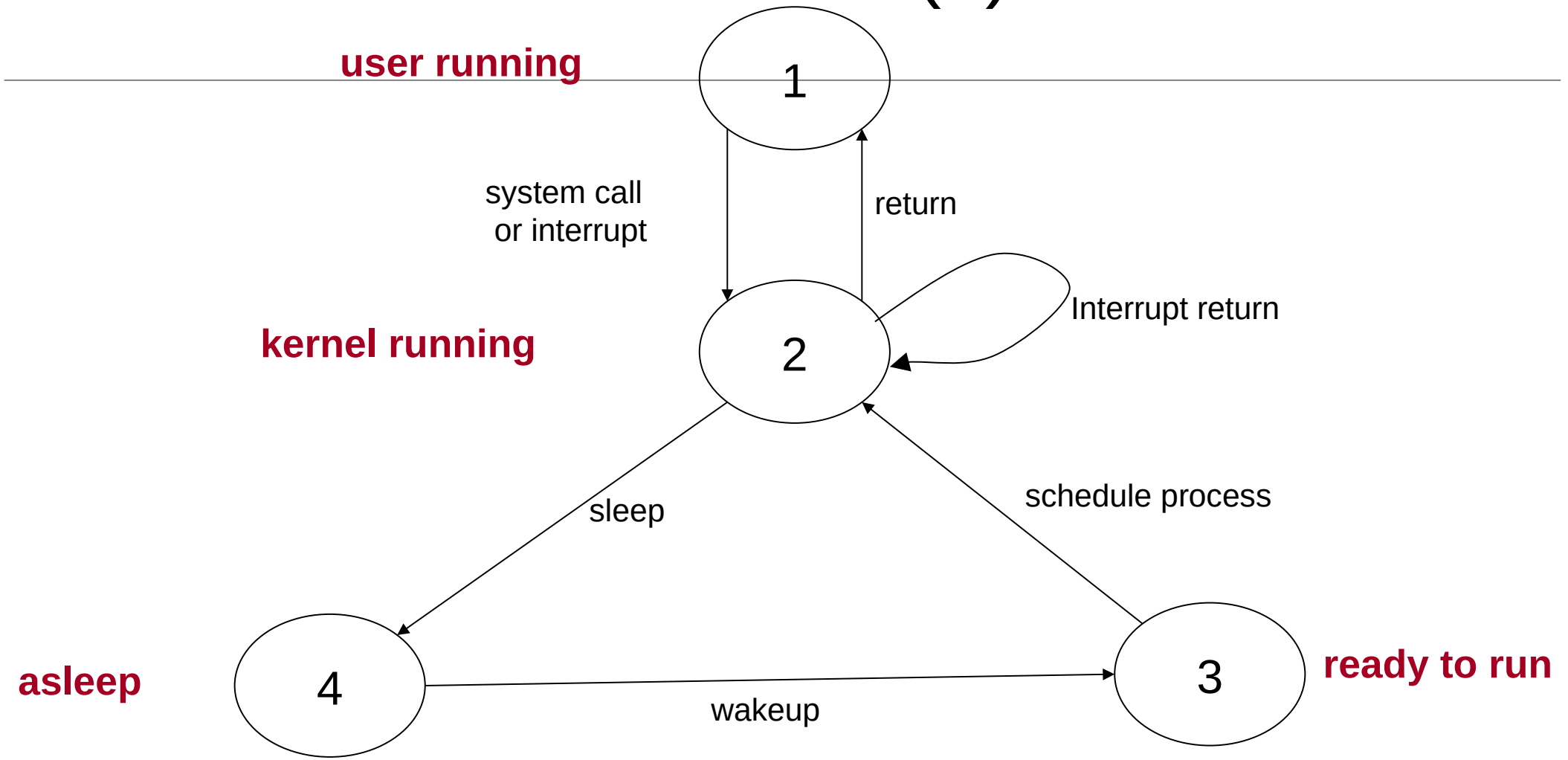
- When moving from user to kernel mode, the kernel saves enough information so that it can later return to user mode and continue execution from where it left off.
- Mode change is not a context switch, just change in mode.

Process States

Process states are:

- The process is running in user mode
- The process is running in kernel mode
- The process is not executing, but it is ready to run as soon as the scheduler chooses it
- The process is sleeping
  - Such as waiting for I/O to complete

Process State Transition(1)



Assumptions about hardware

- Unix OS executes processes into two modes ie user and kernel.
- When a user process makes a system call it goes into the kernel mode , the system call is serviced by the kernel and the control is returned back to the user.
- When a process demands hardware/resources it enters kernel mode.
- When process does not demand hardware and resources it is in user mode.

Process State Transition

- The kernel allows a context switch only when a process moves from the state kernel running to the state asleep
- Process running in kernel mode cannot be preempted by other processes.

Differences between Linux/Unix

- Linux is open-source.Unix is not,though licensed to users,never really free for users. Unix is a propriety software
- Linux has a microkernel architecture. Unix has a monolithic architecture
- Linux kernel can be freely downloaded Unix is expensive/costly
- Linux can be for everyone. Unix mostly for mainframes.
- Linux Kernel is developed by the community. Unix has major distributions :-- Solaris (Oracle), AIX (IBM) & HP-UX Hewlett Packard.
- Linux supports most wider hardware range than Unix.
- Most of the functions/shell commands are supported by both Linux and Unix.

Comparison between Windows,Linux,Unix

WINDOWS	LINUX	UNIX
Developed by Microsoft Cooperation.	Linux kernel developed by Linus Torvalds, in Finland (1991).	Created by **AT&T Bell Laboratories in 1960.**
Windows uses the micro-kernel.	Linux utilizes the monolithic kernel	Unix utilizes the monolithic kernel
Uses a **Graphical User Interface (GUI)**.	Compatibility with the user interface and programming interface	Comes with a **Command Line Interface (CLI)**.
Licensed OS	Open source programming advancement and free of charge operating system	Free and open-source OS
Case sensitivity as an option	The file system in Linux is very case-sensitive.	Fully case-sensitive
Stability has improved greatly in recent years	Linux is Unix-based designed to provide powerful, stable, reliable and easy to use environment.	Very stable to execute.
Supports all of the available hardware	Supports broad range of PC software and hardware	Limited hardware support
It doesn't support multiprocessing.	Linux kernel supports symmetric multi-processing	It supports multiprocessing.
It makes use of the New Technology File System (NTFS) and the File Allocation System (FAT32)	Linux uses Ext, Ext2, Ext3, Ext4, JFS, XFS, btrfs and swap Ext file system.	It uses the Unix File System (UFS), which includes the STD.ERR and STD.IO file systems.
It is less secure than UNIX operating system.	It is more secure than Windows OS	It is more secure because all system updates require explicit user permission.
Windows XP, Vista, Windows 95, Windows 7, 8, 10, and 11.	Ubuntu, RedHat, Solaris, OpenSuse, etc	AIX, HP-UX, BSD, etc

Program to copy the file

```
#include <fcntl.h>
char buffer[2048]; // the uninitialized data is the array buffer
 int version =1; //version should have some initialized data.
main(argc, argv)
int argc;
char *argv[];
{
        int fdold, fdnew;
        if (argc != 3)
         printf("need 2 arguments for copy program\n');
         exit(1);
}
```

```
    fdold=open(argvv[1], O_RDONLY); /* open source file read only */
    if (fdold ==-1)
    {
     printf("cannot open file %s\n", argv[1]);
     exit(1);
    }
    fdnew= creat(argv[2], 0666);
     if (fdnew ==-1) /* create target file rw for all */
    { printf("cannot create file %sn", argv[2]);
    exit(1); }
    copy(fdold, fdnew);  //call to copy subroutine
     exit (0);
}
    copy(old, new)   //definition of copy subroutine
    int old, new;
    {
     int count;
     while ((count=read(old, buffer, sizeof(buffer))) > 0) //The read system call returns the number of bytes read,
    returning 0 when it reaches the end of file.
    write(new, buffer, count); //write the data to new file
    }
```

Explanation of program

- fcntl.h -The header in the C POSIX library for the C programming language that **contains constructs that refer to file control**, e.g. opening a file, retrieving and changing the permissions of file, locking a file for edit, etc.
- The first parameter, argc (argument count) is an integer that indicates how many arguments were entered on the command line when the program was started. The second parameter, argv (argument vector), is an array of pointers to arrays of character objects.
- argv[0] points to the character string copy (the program name is conventionally the 0th parameter)
- argv[1] points to the character string oldfile, and argv[2]points to the character string newfile

Explanation cntd….

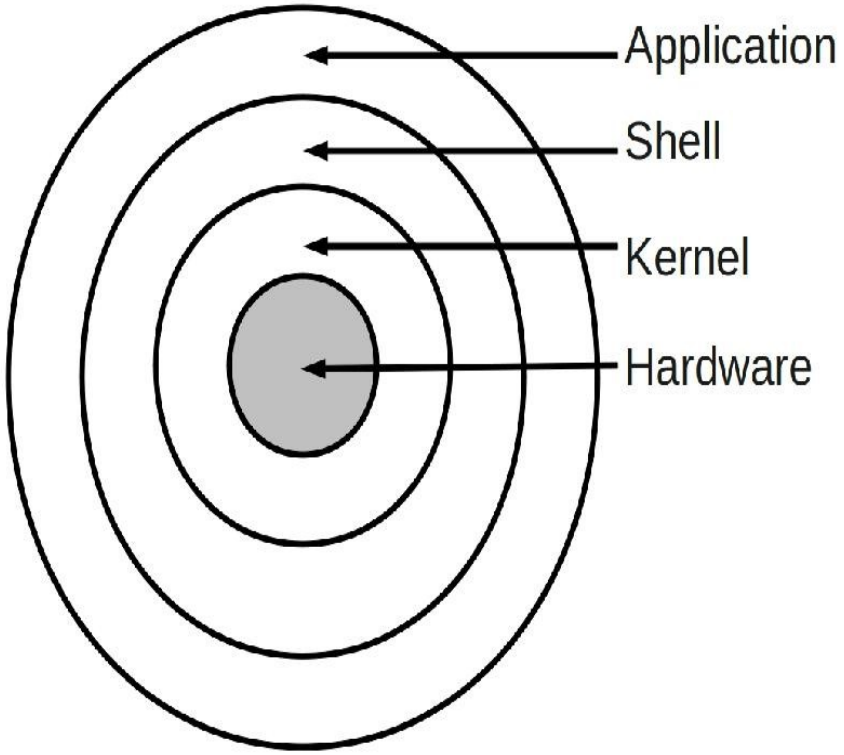
- The permission modes on the newly created file will be 0666 (octal), allowing all users access to the file for reading and writing.
- The open and creat system calls return an integer called a file descriptor, which the program uses for subsequent references to the files.

What is Shell ?

- The "**Shell**" is simply ***another program*** on top of the kernel which provides a basic human-OS interface.

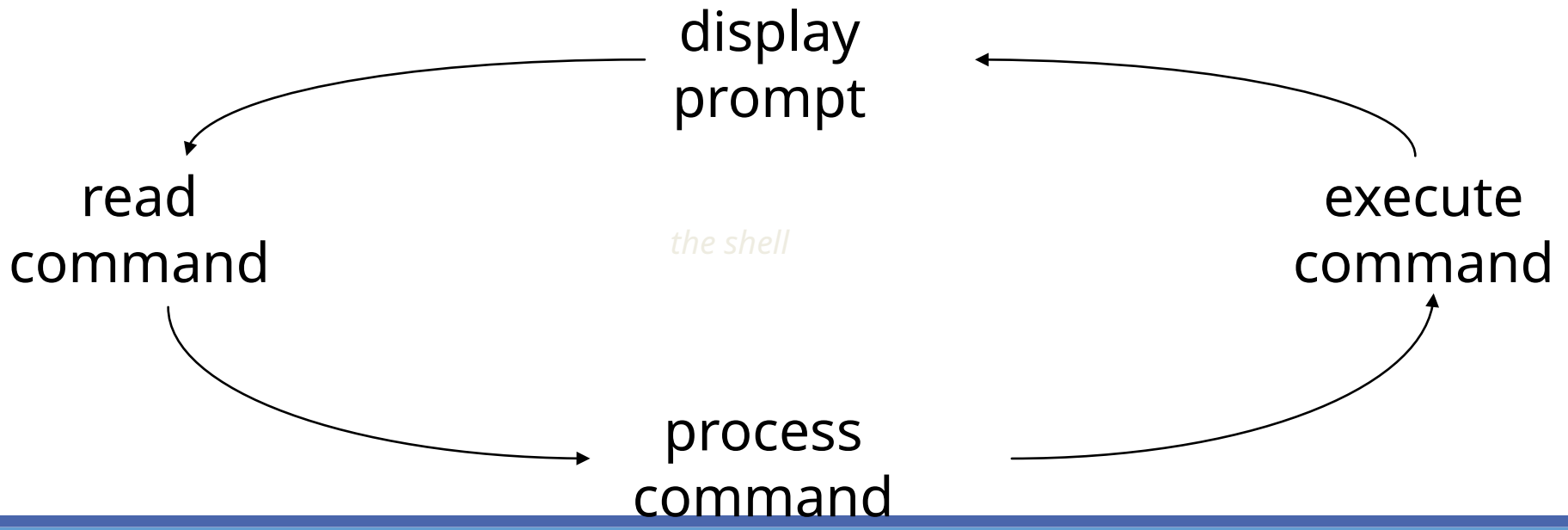
  It is a command interpreter
  Enables users to run services provided by the UNIX OS

- Functionality:
  - Command Interpreter
  - Interface
  - Programming language



The Shell

- The UNIX user interface is called the *shell*.
- The shell does 4 jobs repeatedly:



Most Commonly Used Shells

/bin/csh	-- C shell
/bin/tcsh	-- Enhanced C Shell
/bin/sh	-- The Bourne Shell / POSIX shell
/bin/ksh	-- Korn shell
/bin/bash	-- Korn shell clone, from GNU

All Linux versions use the Bash shell (Bourne Again Shell) as the default shell

➢ To find all available shells in your system type following command

$ cat /etc/shells

($ means terminal prompt)

➢ To Check default Shell of your system

$ echo $SHELL

(SHELL is a pre-defined variable)

What is Shell Script ?

- A shell script is a series of command(s) stored in plain text file.
- **Why to Write Shell Script ?**

  Shell script can take input from user or file and output them on screen.

  Useful to create our own commands. Save lots of time.

  To automate some task of daily life.

  System Administration part can be also automated.

MIT-WPU
॥ विश्वशान्तिर्ध्रुवं ध्रुवा ॥

Practical examples where shell scripting actively used:

1.Monitoring your Linux system.
2.Data backup.
3.Find out what processes are eating up your system resources.
4.Find out available and free memory.
5.Find out all logged in users and what they are doing.
6.Find out if all necessary network services are running or not.
And many more…..

Create a script

➢Use gedit as text editors for shell scripting
Example:
$ gedit hello
and type the following inside it:
#!/bin/bash
echo “Hello World”
The first line tells Linux to use the bash interpreter to run this script as hello.sh.
bash comments start with a hash mark (#)

➢**To prepare the file for running, just turn on its execute bit**

➢**Syntax to setup executable permission:**
$ chmod +x your-script-name.
$ chmod 755 your-script-name.

Run a script (execute a script)

➢ **Different ways to run the script :**

1.bash your-script-name

2.sh your-script-name

3../your-script-na

➢ **Example :**

$ bash hello

$ sh hello

$ ./hello

Example (hello script)

#!/bin/bash

echo “Hello, world!”

Output:

$ chmod +x hello

$ ./hello

Hello, world!

Variables in Shell

- Variables are global within a script
- Types of variable:

System variables - Created and maintained by Linux itself. This type of variable defined in CAPITAL LETTERS.

User defined variables (UDV) – Created and maintained by user. This type of variable defined in lower letters. To define UDV use following syntax:

variable name=value

Examples: **(You must *use $ followed by variable name whe***

```
vech=Bus
n=10
echo $vech
echo $n
```

```
mylang="Pennsylvania Dutch"
echo "I speak $mylang."
I speak Pennsylvania Dutch.
 echo 'I speak $mylang.'
I speak $mylang.
```

Input / Output

- Use the **read** command to prompt for input from user

Syntax:

read var_name

- Use of **echo and printf** command to

Syntax:

echo “Message” OR echo $ var_name

Example :

```
$ echo "Your first name please:"
$ read fname
$ echo "Hello $fname, Lets be friend!“
```

Bash Comparison Operator

Elementary bash comparison operators

String	Numeric	True if
x = y	x -eq y	x is equal to y
x != y	x -ne y	x is not equal to y
x < y	x -lt y	x is less than y
x <= y	x -le y	x is less than or equal to y
x > y	x -gt y	x is greater than y
x >= y	x -ge y	x is greater than or equal to y
-n x	–	x is not null
-z x	–	x is null

Meaning	Numeric	String
Greater than	-gt	
Greater than or equal	-ge	
Less than	-lt	
Less than or equal	-le	
Equal	-eg	= or ==
Not equal	-ne	!=
str1 is less than str2		str1 < str2
str1 is greater str2		str1 > str2
String length is greater than zero		-n str
String length is zero		-z str

Conditional Control Statement: if

Syntax: if….fi

```
if condition
then
command1            #if condition is true
Fi
```

Syntax: if…else...fi

```
if condition
 then
command1            # if condition is true
else
Command2            # if condition is flase
fi
```

Example:

```
$ gedit myscript.sh
read choice
if [ $choice -gt 0 ]
then
echo "$choice number is positive"
else
echo "$choice number is
negative"
fi
```

Loops in Shell Scripts

for Loop

Syntax:

```
for { variable name } in { list }
do
    #execute one for each item in the list until the list is
        not finished and repeat all statement between do
        and done
done
```

Example :

```
for i in 1 2 3 4 5
do
   echo "Welcome $i times"
done
```

while loop

Syntax :

```
while [ condition ]          #space required
do
        command1 command2
command3 .. ....
done
```

Example :

```
i=1; n=1
while [  $i -le 10  ]
do
        echo "$n * $i =" `expr $i \* $n`
        i=`expr $i + 1`
done
```

case Statement in Shell Scripts

Syntax:

case *$variable-name* ***in***
pattern1) command…..;;
pattern2) command…..;;
pattern N) command….;;
.
**) command ;;* #default case
esac

Example :

```
read var
case $var in
1) echo “One”;;
2) echo “Two”;;
3) echo “Three”;;
4) echo “Four”;;
*) echo "Sorry, it is bigger than Four";;
esac
```

Functions in Shell Scripts

Function is series of instruction/commands.

Function performs particular activity in shell.

Syntax:

function-name ( )

{

Function body

}

Example :

```
today()
{
    echo “Today is `date`”
    return
}

today
```

Special Files

/home - all users' home directories are stored here

/bin, **/usr/bin** - system commands

/sbin, **/usr/sbin** - commands used by sysadmins

/etc - all sorts of configuration files

/var - logs, spool directories etc.

/dev - device files

/proc - special system files