

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

COURSE FILE

LINUX PROGRAMMING



(ESTD - 1999)

Vidya Jyothi Institute of Technology

**(Accredited by NAAC & NBA, Approved By A.I.C.T.E., New Delhi,
Permanently Affiliated to J.N.T. University, Hyderabad)**

(Aziz Nagar, C.B.Post, Hyderabad -500075)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

REGULATION: R15

BATCH: 2017-2021

ACADEMIC YEAR: 2019-20

PROGRAM: B.TECH (COMPUTER SCIENCE AND ENGINEERING)

YEAR/SEM: III/I

COURSE NAME LINUX PROGRAMMING

COURSE CODE: A15512

PREREQUISITE: OPERATING SYSTEM

COURSE COORDINATOR: M.VIJAYA

COURSE INSTRUCTORS:

K.SAMATHA

Dr.K. RANGA RAO

P.LAKSHMI PRIYA

COURSE FILE INDEX

S.NO.	DESCRIPTION
1	Syllabus
2	Text Book & Other References
3	Time Table
4	Program Outcomes (PO's) ,Program Specific Outcomes (PSO's) &PEO's
5	Mapping Of Course Outcomes (CO's) With Program Outcomes (PO's) & Program Specific Outcomes (PSO's)
6	Academic Calendar
7	Course Schedule
8	Lesson Plan
9	Assignment Questions
10	Mid Question Papers I & II
11	Unit Wise Questions
12	Minutes of Course Review Meeting
13	Lecture Notes
14	Power Point Presentation
15	Semester End Question Papers
16	Extra Topics Delivered (if any)
17	Innovations In Teaching and Learning
18	Assessment Sheet – Co Wise (Direct Attainment)
19	Course End Survey Form

Syllabus

LINUX PROGRAMMING

Course Objectives:

- To understand the Linux utilities, sed and awk concepts to solve problems.
- To implement in C some standard Linux utilities such as ls, mv, cp etc. using system calls.
- To understand process concepts and Interprocess communication in Linux.

Course Outcomes:

- To understand and make effective use of Linux utilities and Shell scripting language (bash) to solve problems.
- To develop the skills necessary for systems programming including file system programming, process and signal management.
- To apply basic skills of inter process communication
- To develop the basic skills required to write network programs using Sockets

UNIT - I:

Linux Utilities: File handling utilities. Security by file permissions. Process utilities, Disk utilities, Networking commands, Filters, Text processing utilities and Backup utilities.

Sed-Scripts, Operation, Addresses, Commands, Applications, awk-Execution, Fields and Records, scripts, operations, patterns, actions, functions, using system commands in awk.

UNIT - II:

Shell programming with Bourne again shell(bash): Introduction, shell responsibilities, pipes and Redirection, here documents, running a shell script, the shell as a programming language, shell meta characters, file name substitution, shell variables, command substitution, shell commands, the environment, quoting, test command, control structures, arithmetic in shell, shell script examples, interrupt processing functions, debugging shell scripts.

UNIT - III:

Files : File Concept, File types, File System Structure, Inodes, File Attributes, Library Functions, kernel support for files, system calls for file I/O operations- open, create, read, write, close.

Directories: Creating, removing and changing Directories -mkdir, rmdir, chdir, obtaining current working directory. Scanning Directories- opendir, readdir, closedir, rewinddir functions.

UNIT - IV:

Process: Process Concept, process identification, process control - process creation, waiting for a process, process termination, Kernel support for process, zombie process, orphan process, **Process APIs.** **Signals** - Introduction to signals, Signal generation and handling, Kernel support for signals, Signal function, unreliable signals, reliable signals, kill, raise, alarm, pause, abort, sleep functions.

UNIT - V:

Inter Process Communication: Introduction to IPC, IPC between processes on a single computer system, IPC between processes on different systems, pipes-creation, IPC between related processes using unnamed pipes, FIFOs- creation, IPC between unrelated processes using FIFOs (Named pipes), differences between unnamed and named pipes, open and close library functions. Message Queues- APIs for message queues, Semaphores-APIs for semaphores Shared Memory: APIs for shared memory. Sockets: Introduction to Sockets, Socket address structures, Socket system calls for connection oriented protocol and connectionless protocol.

Text Books & Other References

Text Books, Reference Books

Text Books	
1	Unix Concepts and Applications, 4th Edition, Sumitabha Das, TMH
2	Unix and shell Programming, B.A. Forouzen and R.F. Gilberg, Cengage learning.
3	Unix Network Programming, W.R.Stevens, PHI
Suggested / Reference Books	
1	Unix System Programming using C++, T.Chan, PHI.
2	Beginning Linux Programming, 4th Edition, N. Mathew, R. Stones, Wrox, Wiley India Edition.
3	Unix for programmers and users, 3rd Edition, Graham Glass, King Ables, Pearson.
4	Unix shell Programming, S. G. Kochan and P. Wood, 3rd edition, Pearson Education
5	Shell Scripting, S. Parker, Wiley India Pvt. Ltd.
6	C Programming Language, Kernighan and Ritchie, PHI.
Other Resources	
1	iare.ac.in/sites/default/files/lecture_notes/LP
2	https://www.smartzworld.com/notes/linux-programming-lp/
3	vjit.ac.in/it/study-material/

Time table

VIDYA JYOTHI INSTITUTE OF TECHNOLOGY
Department of Computer Science & Engineering

Sec: **CSE-A**

Year/Sem: **III -I**

W.E.F:17/06/2019

ROOM NO: **N202**

DAY	9.00 – 10.00	10.00 – 11.00	11.00 – 12.00	12.00 – 12.45	12.45 – 1.45	1.45 – 2.45	2.45 – 3.45
MON	PE	OS	LP(T)	LUNCH	ACS LAB		
TUE	CN	OE			LP	FLAT	OS(T)
WED	OS	OE			LP	CN	FLAT(T)
THU	LP	CN	OS		PE	FLAT	MC-III
FRI	LP/CN/OS LAB				MC-III	CN	PE(T)
SAT	PE	FLAT	OS		FLAT	LP	CN(T)

Subject

Name of the Faculty

LP	Linux Programming	Ms.M.Vijaya
CN	Computer Networks	Ms.G Surekha
OS	Operating Systems	Ms.T.Maanasa
FLAT	Formal Languages and Automata Theory	Mr.B.Thikkana
HCI/PPL	Human Computer Interaction/Principles of Programming Language	MR.Zeeshan
OS & CN LAB	Operating Systems & Computer Networks lab through LINUX.	Ms.M.Vijaya/ Ms.T.Maanasa / Ms.G Surekha
ACS LAB	Advanced Communication Skills Lab	Ms.Indira Priyadarshini
MC - III	Quantitative Methods & Logical Reasoning	Vijay Babu

Class Incharge
 III YEAR Coordinator

Ms.T.Maanasa
 Ms.M.Vijaya

Time Table I/c

H.O.D

**Program Outcomes (PO's), Program
Specific Outcomes (PSO's) &
Program Educational Objectives
(PEO's)**

Programme Outcomes (PO's)

1. **Engineering knowledge:** Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization for the solution of complex engineering problems.
2. **Problem analysis:** Identify, formulate, research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
3. **Design/development of solutions:** Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for public health and safety, and cultural, societal, and environmental considerations.
4. **Conduct investigations of complex problems:** Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
5. **Modern tool usage:** Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools, including prediction and modeling to complex engineering activities, with an understanding of the limitations.
6. **The engineer and society:** Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal, and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
7. **Environment and sustainability:** Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
8. **Ethics:** Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
9. **Individual and team work:** Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
10. **Communication:** Communicate effectively on complex engineering activities with the engineering community and with the society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
11. **Project management and finance:** Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
12. **Life-long learning:** Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

Program Specific Outcomes

PSO 1: The ability to design and develop Algorithms to provide optimized solutions for societal needs

PSO 2: Apply standard approaches and practices in Software Project Development through trending technologies

Program Educational Objectives (PEO's)

PEO1: Enhance the employability of the graduate in software industries/Public sector/Research organizations

PEO2: Acquire analytical and computational abilities to pursue higher studies for professional growth

PEO3: Work in multidisciplinary project teams with effective communication skills and leadership qualities

PEO4: Develop professional ethics among the students and promote entrepreneurial abilities

**Mapping Of Course
Outcomes (CO's) with
Program Outcomes (PO's) &
Program Specific Outcomes
(PSO's)**



VIDYA JYOTHI INSTITUTE OF TECHNOLOGY
Department of Computer Science & Engineering

Year & Sem: III Year I Sem

Course name: Linux Programming

Course Code: A15512

Regulation: R15

COURSE OUTCOMES:

After completing this course the student must demonstrate the knowledge and ability to	
CO1	Understand and make effective use of Linux utilities.
CO2	Able to write shell scripts to solve the problems.
CO3	Develop the skills necessary for file system and directory handling.
CO4	Learn the concepts of process and signal system calls.
CO5	Implement inter process communication mechanisms.

CO -PO MAPPING:

	PO 1	PO 2	PO 3	PO 4	PO 5	PO 6	PO 7	PO 8	PO 9	PO 10	PO 11	PO 12
CO 1	3	3	2	2	2	-	3	3	-	1	2	2
CO 2	3	3	3	2	2	-	-	3	3	2	3	2
CO 3	3	3	3	3	1	-	-	3	-	2	2	2
CO 4	3	3	3	2	2	-	-	3	3	2	3	2
CO 5	3	3	3	3	2	3	1	3	-		2	2
Avg	3	3	2.8	2.4	1.8	3	2	3	3	1.75	2.4	2

CO - PSO MAPPING:

	PSO1	PSO2
CO1	3	2
CO2	3	2
CO3	3	2
CO4	3	2
CO5	3	2
Avg	3	2

Signature of the Course Coordinator.

Sign. of HOD

Academic Calendar



Vidya Jyothi Institute of Technology (Autonomous)

(Accredited by NAAC & NBA, Approved by AICTE, New Delhi. Formerly Affiliated to JNTU, Hyderabad)
(Aziz Nagar, C.B. Road, Hyderabad - 500075)

II B.Tech I & II Semester Academic Calendar for the Academic Year 2019-20

II YEAR I SEMESTER		Commencement of Class Work 17.06.2019	
	From	To	Duration
I Spell of Instruction	17.06.2019	10.08.2019	8 WEEKS
I Mid Examinations	13.08.2019	17.08.2019	4 DAYS
II Spell of Instruction	19.08.2019	05.10.2019	7 WEEKS
Dussehra Holidays	07.10.2019	12.10.2019	1 WEEK
II Spell of Instruction Continuation	14.10.2019	19.10.2019	1 WEEK
II Mid Examinations	21.10.2019	24.10.2019	4 DAYS
Practical Examinations	25.10.2019	29.10.2019	4 DAYS
Betterment Examinations	30.10.2019	01.11.2019	3 DAYS
End Semester Examinations	02.11.2019	18.11.2019	2 WEEKS
Supplementary Examinations	19.11.2019	04.12.2019	2 WEEKS
Mid-II Examinations (For Lateral Entry)	18.11.2019	21.11.2019	4 DAYS
Practical Examinations (For Lateral Entry)	22.11.2019	26.11.2019	4 DAYS
Betterment Examinations (For Lateral Entry)	27.11.2019	30.11.2019	4 DAYS
II YEAR II SEMESTER		Commencement of Class Work 02.12.2019	
I Spell of Instruction	02.12.2019	10.01.2020	6 WEEKS
Pongal Holidays	11.01.2020	15.01.2020	5 DAYS
Technical/Sports fest	16.01.2020	18.01.2020	3 DAYS
I Spell of Instruction Continuation	20.01.2020	01.02.2020	2 WEEKS
I Mid Examinations	03.02.2020	08.02.2020	1 WEEK
II Spell of Instruction	10.02.2020	04.04.2020	8 WEEKS
II Mid Examinations	06.04.2020	09.04.2020	4 DAYS
Practical Examinations	13.04.2020	17.04.2020	4 DAYS
Betterment Examinations	18.04.2020	22.04.2020	4 DAYS
End Semester Examinations	23.04.2020	08.05.2020	2 WEEKS
Supplementary Examinations	11.05.2020	23.05.2020	2 WEEKS
Commencement of classes will be from 15.06.2020			


DIRECTOR

Course Schedule

Distribution of Hours in Unit – Wise

Unit	Topic	Book1	Book2	Book3	Total No. of Hours
I	Linux Utilities	Ch1-1.1-1.11 Ch2-2.1-2.12 Ch3-3.1-3.24			10
II	Shell programming with Bourne again shell(bash)		Ch5-5.1-5.15		14
III	Files and Directories	Ch3-3.1-3.4	Ch3-3.1-3.7		10
IV	Process Signals	Ch7-7.1-7.9			12
V	Inter Process Communication			Ch2,Ch4, Ch6,Ch10, Ch12,Ch13	16
Total contact classes for syllabus coverage					62
Assignment Tests : 02(Before Mid1 & Mid2 Examination)					

Number of hours available in Semester/ Year: 64

```

if[$count -eq 10]
then
echo "condition is true"
else
echo "condition is false"
fi
8. Write a Shell script to find factorial of a given integer
Ans.

```

```

# !/bin/bash
echo "enter a number"
read num
fact=1
while [ $num -ge 1 ]
do
fact=`expr $fact * $num`
num=`expr $num - 1`
done
echo "factorial of $n is $fact"

```

Assignment Operators

There are following assignment operators supported by LINUX PROGRAMMING

Show Examples

Op	Description	Example
=	Simple assignment operator. Assigns values from right side operands to left side operand.	C = A + B will assign value of A + B into C
+=	Add AND assignment operator. It adds right operand to the left operand and assign the result to left operand.	C += A is equivalent to C = C + A
-=	Subtract AND assignment operator. It subtracts right operand from the left operand and assign the result to left operand.	C -= A is equivalent to C = C - A
*=	Multiply AND assignment operator. It multiplies right operand with the left operand and assign the result to left operand.	C *= A is equivalent to C = C * A

- != Checks if the value of two operands are equal or not, if values are not equal then cond (A != B) is becomes true.
- > Checks if the value of left operand is greater than the value of right operand, if yes the (A > B) is condition becomes true.
- < Checks if the value of left operand is less than the value of right operand, if yes then (A < B) is condition becomes true.
- >= Checks if the value of left operand is greater than or equal to the value of right operand (A >= B) is yes then condition becomes true.
- <= Checks if the value of left operand is less than or equal to the value of right operand, i (A <= B) is then condition becomes true.

Syntax

```

if [ expression 1 ]
then
    Statement(s) to be executed if expression 1 is true
elif [ expression 2 ]
then
    Statement(s) to be executed if expression 2 is true
elif [ expression 3 ]
then
    Statement(s) to be executed if expression 3 is true
else
    Statement(s) to be executed if no expression is true
fi

```

This code is just a series of *if* statements, where each *if* is part of the *else* clause of the previous statement. Here statement(s) are executed based on the true condition, if none of the condition is true then *else* block is executed

The case...esac Statement

You can use multiple **if...elif** statements to perform a multiway branch. However, this is not always the best solution, especially when all of the branches depend on the value of a single variable.

Unix Shell supports **case...esac** statement which handles exactly this situation, and it does so more efficiently than repeated **if...elif** statements.

There is only one form of **case...esac** statement which has been described in detail here

- case...esac statement

The **case...esac** statement in the Unix shell is very similar to the **switch...case** statement we have in other programming languages like C or C++ and PERL, etc.

In this chapter, we will discuss shell loops in Unix. A loop is a powerful programming tool that enables you to execute a set of commands repeatedly. In this chapter, we will examine the following types of loops available to shell programmers

- The while loop
- The for loop
- The until loop
- The select loop

You will use different loops based on the situation. For example, the **while** loop executes the given commands until the given condition remains true; the **until** loop executes until a given condition becomes true.

Once you have good programming practice you will gain the expertise and thereby, start using appropriate loop based on the situation. Here, **while** and **for** loops are available in most of the other programming languages like C, C++ and PERL, etc.

Nesting Loops

All the loops support nesting concept which means you can put one loop inside another similar one or different loops. This nesting can go up to unlimited number of times based on your requirement.

Here is an example of nesting **while** loop. The other loops can be nested based on the programming requirement in a similar way

Nesting while Loops

It is possible to use a while loop as part of the body of another while loop.

Syntax

```
while command1 ; # this is loop1, the outer loop
do
  Statement(s) to be executed if command1 is true
```

- The **continue** statement

The infinite Loop

All the loops have a limited life and they come out once the condition is false or true depending on the loop.

A loop may continue forever if the required condition is not met. A loop that executes forever without terminating executes for an infinite number of times. For this reason, such loops are called infinite loops.

Example

Here is a simple example that uses the **while** loop to display the numbers zero to nine –

```
#!/bin/sh
```

```
a=10
```

```
until [ $a -lt 10 ]
```

```
do
```

```
    echo $a
```

```
    a=expr $a + 1`
```

```
done
```

This loop continues forever because *a* is always greater than or equal to 10 and it is never less than 10.

The break Statement

The **break** statement is used to terminate the execution of the entire loop, after completing the execution of all of the lines of code up to the break statement. It then steps down to the code following the end of the loop.

Syntax

The following **break** statement is used to come out of a loop

```
break
```

The break command can also be used to exit from a nested loop using this format –

```
break n
```

Here *n* specifies the *n*th enclosing loop to the exit from

Example

Here is a simple example which shows that loop terminates as soon as *a* becomes 5 –

```
#!/bin/sh
```

```
a=0
```

This statement is useful when an error has occurred but you want to try to execute the next iteration of the loop.

Syntax

`continue`

Like with the `break` statement, an integer argument can be given to the `continue` command to skip commands from nested loops.

`continue n`

Here `n` specifies the n^{th} enclosing loop to continue from.

FIFO file:-

First in first out queue is a type of file used for Inter Process Communication(IPC) between the processes.

The FIFO file is also called as the pipe.

Pipes are tools that allow two or more system processes to communicate with each other using a file that acts as a pipe between them.

Socket file:-

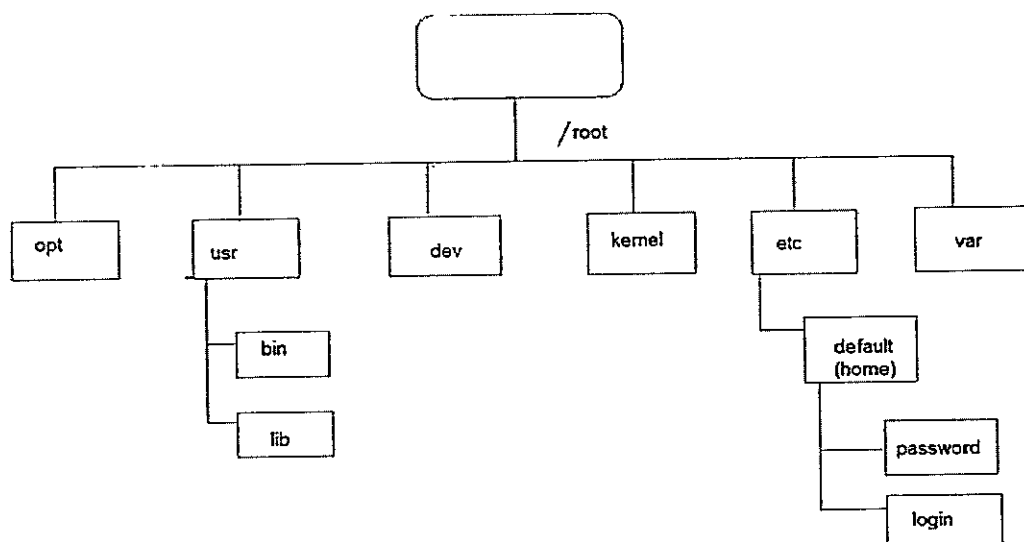
A socket is a type of file used for network communication between the processes.

Sockets are also tools used for inter process communication. The difference between sockets and pipes is that sockets will facilitate communication between processes running on different systems, or over the network.

Symbolic link:-

It is a type of file that contains the address or path-name of another.

A Symbolic link is a tool used for having multiple filenames that reference a single file on a physical disk. They appear in a file system just like an ordinary file or a directory.

File system structure

In UNIX
file

system,

each directory has a parent. The root directory has no parent and it is represented by '/' and which contain more files, sub-directories etc.

root:-

root is a directory file and it has many number of sub-directories under it. These sub-directories have more sub-directories and other files under them.

bin:-

10. Number of links

In UNIX, file system has a inode table which keeps track of all files.

Each entry of inode table is an inode record which contains all attributes of a file.

File Attributes

The attributes of a file can be determined using `stat` and `fstat` system calls and here we are using a header `#include <sys/stat.h>`

Every file has many file attributes such as:-

ATTRIBUTES	MEANING
File type	Type of file
Access permission	File access permission for owner, group, others
Hard link count	Number of hard links in a file
UID	User ID or Owner ID
GID	The file group ID
File size	The size of a file in bytes
Last access time	The time at which the file was last accessed
Last modify time	The time at which the file was last modified
Last change time	The time the file access permission, UID, hard link was last changed
Inode number	The system inode number of a file
File system ID	The file system ID where the file is stored

Library functions

UNIX system provides a large number of C functions as libraries. Some of those functions will be used frequently for performing operations, while the others are very specialized in their applications.

FUNCTIONS	DESCRIPTION
<code>abs</code>	Integer absolute value
<code>ctime</code>	Convert date and time
<code>fopen</code>	Open a stream
<code>printf</code>	Formatted output
<code>fputc</code>	Put a character or a word on stream
<code>getcwd</code>	Get current working directory path name
<code>strcat</code>	String concatenation

is NULL, fflush() flushes all open output streams.

Syntax:- The function prototype of fflush() is

```
#include<stdio.h>
int fflush(FILE* stream);
```

4. fseek():-

The fseek() function sets the file position indicator for the stream pointed to by the stream.

Syntax:- The function prototype of fseek() is

```
#include<stdio.h>
int fseek(FILE*stream, long offset, int whence);
```

5. fgetc():-

This function reads the next character as an unsigned char from the stream and returns its value, converted to int.

Syntax:- The function prototype of fgetc() is

```
#include<stdio.h>
int fgetc(FILE *stream);
```

6. getc():-

The getc() function is equivalent to fgetc() function except that it can be implemented as a macro which evaluate stream more than once.

Syntax:- The function prototype of getc() is

```
#include<stdio.h>
int getc(FILE * stream);
```

7. getchar():-

The getchar() function is equivalent to fgetc() with stdin as the value of stream argument.

Syntax:- The function prototype of getchar() is

```
#include<stdio.h>
int getchar(void);
```

8. fputc():-

The fputc() function is declared in the header stdio.h. The fputc() converts the character c to type unsigned char and will write it to the stream.

End of file is returned if a write error occurs, otherwise the character c is returned.

Syntax:- The function prototype of fputc() is

```
#include<stdio.h>
int fputc(int c,FILE* stream);
```

9. putc():-

The putc() function is same as the fputc() function except that most system implements it as macro, making it faster.

Syntax:- The function prototype of putc() function is

```
#include<stdio.h>
int putc(int c, FILE* stream);
```

10. putchar():-

The putchar() function is equivalent to fputc() with stdout as a value of the stream argument.

Syntax:- The function prototype of putchar() is

Formatted Input functions

The following are the formatted input functions:-

scanf()
fscanf()
sscanf()

1. scanf():-

The scanf() function reads formatted input from the stream stdin under the control of the template stream template.

The optional arguments are pointers to place which receive the resulting values.

Syntax:- #include<stdio.h>
int scanf(const char* template);

2. fscanf():-

The fscanf() function is just like scanf() except that the input is read from the stream instead of stdin.

Syntax:- #include<stdio.h>
int fscanf(FILE * stream, const char * template);



3. sscanf():-

The sscanf() function is also same as scanf() except that the characters are taken from the NULL terminated stream s instead from stream.

Reaching the end of stream is treated as end of file condition

Syntax:- #include<stdio.h>
int sscanf(char *s, const char*template);

System calls

System calls can be defined as the signals generated to kernel for performing specific tasks.

System calls are the programmers functional interface through which programs can directly interact with heart of the UNIX(kernel). To make use of its services such as file creation each system call has a predefined task to be performed for there are system calls to open a file, close a file, execute a process, write a data to a file, display the time of the day & so on.



Low Level File Access System Calls:-

1. **Open():** This function is used to open or create a file, after a file is created any process can call the open() function to get the file descriptor to refer the file.

The function prototype of open() is

```
#include <Sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <error.h>
int open(const char *path name, int flag);
      or
      (int mode (or) mode tmode)
```

Standard libraries and system calls provide complete control over the creation and maintenance of files and directories.

Some of the functions used are:-

chmod()
chown()
unlink()
link()
mkdir()
rmdir()
chdir()

1. chmod():-

The chmod() i.e change mode system call helps to modify the permission flag.

Syntax:- The function prototype of chmod() is

```
#include<sys/types.h>
#include<sys/stat.h>
#include<fcntl.h>
#include<error.h>
#include<unistd.h>
int chmod(const char* filename, int mode);
```

2. chown():-

The chown() system call alters the owner or group of a file.

chown() system call helps in modifying the owner and group ID's of a file. In this on success, it returns a 0, on error returns -1.

Syntax:- The function prototype of chown() is

```
#include<sys/types.h>
#include<sys/stat.h>
#include<fcntl.h>
#include<error.h>
#include<unistd.h>
int chown(const char* filename, uid_t owner_id, gid_t groupid);
```

3. unlink():-

The unlink() system call deletes the hard link of a file from the directory

Syntax:- The function prototype of unlink() is

```
#include<sys/types.h>
#include<sys/stat.h>
#include<fcntl.h>
#include<error.h>
#include<unistd.h>
int unlink(const char* filename);
```

4. link():-

The link() system call allows the user to create hard link to the existing path.

Syntax:- The function prototype of link() is

```
#include<sys/types.h>
#include<sys/stat.h>
#include<fcntl.h>
#include<error.h>
```

UNIT-4

PROCESS

PROCESS CONCEPT

- A process is a instance of a program under execution.
- It is an entity that provides the execution environment for a program & runs the program.
- It has its own address space & comprises a control point.
- CPU runs only 1 process at a time.
- It uses various system resources like memory devices etc.
- It also request the services from the system which are performed by the Linux kernel.
- Each linux process has a parent & 1 or more children.
- It is created by calling the fork() or vfork() system call.
- It may execute 1 or more programs by invoking exec system.
- The lifetime of a process begins when it is created and ends when the process terminates by calling exit system call.
- When the process having one or more children to terminate, all its processes becomes orphans and are inherited by init process.
- Each and every process has a unique non-negative identifier called the process ID.

PROCESS IDENTIFICATION

- A *PID* stands for **process identification number**.
- It is an identification number that is automatically assigned to each process when it is created on a unix-like operating system.
- It is needed in order to terminate a frozen program with the kill command.
- The process init is the only process that will always have the same PID on any session & that PID is 1.
- A very large PID does not necessarily mean that there are anywhere near that many processes on a system
- The default maximum value of PID is 32,767.
- There is a numbered directory in/proc corresponding to each PID currently on the system.
- Each process in linux has a unique id and has a parent.
- We can get the process id of a running process and its parent's process id using following functions,
 1. pid_t getpid()
 2. pid_t getppid()
- Here PID represents process id and PPID represents parent process id.

PROCESS CONTROL

The concept of process control deals with 3 types of independent mechanism namely :

- Once the child and parent process are created, they both start executing the statement.
- During the execution the child process gives few features of the parents like data space, root directory, session ID, process group ID etc.
- Ex: to demonstrate the process execution consider the following function:

```
#include<sys/types.h>
#include "outhdr.h"
intvar = 6;
char store[ ] = "iron armour";
main(void)
{
int v = 88;
pid_t pid;
if(write(STDOUT_FILENO, store, sizeof(store)-1 != sizeof(store)-1))
err_sys("there is error in writing");
if((pid= fork())<0)
err_sys("error while using fork");
else if(pid==0)
{
Var++;
V++;
}
else
sleep(2);
exit(0);
}
```

3.Process Termination

- A process is an instance of an executing program.
- A process performs certain tasks and terminates. The end of the process is called as termination.
- A process can terminate normally or abnormally.

a) Normal termination of a process

A process is said to be terminated normally if,

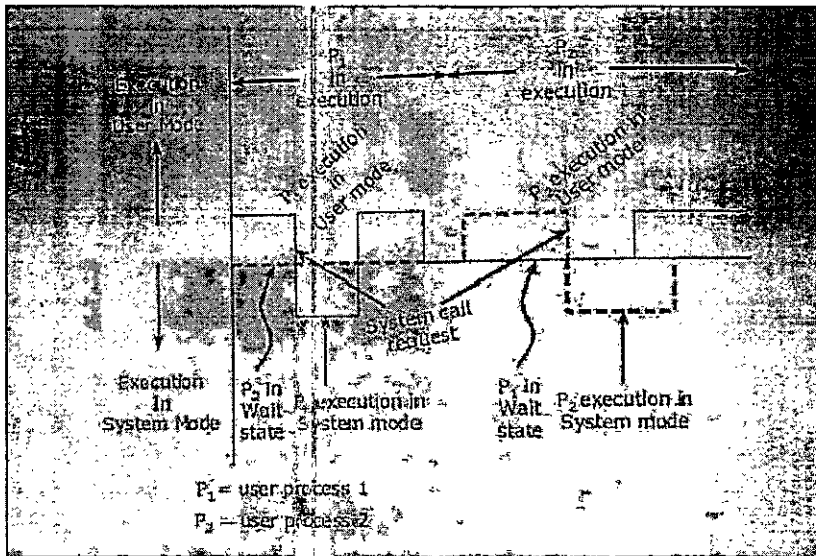
- It executes a return statement in the function. This is similar to calling the exit function.
- It calls the exit functions. This function flushes the buffer and then returns to the kernel.
- It calls the _exit functions. This function immediately returns to the kernel & flushing buffers.

It is called by the exit function internally.

b) Abnormal termination of a process

A process is said to be terminated abnormally if,

- It calls abort function which generates the SIGABRT signal.



ZOMBIE PROCESS

- When a process terminates & its parent does not wait for it then it is called a **Zombie Process**.
- It remains in zombie state until it is cleaned up by its parent.
- It does not have any code, data or stack.
- It still uses the system's fixed-size process called **proc structure**.
- Proc structure maintains the information process ID, the terminated status of the process & the amount of CPU time taken by the process.
- Too many zombies reside in a system and they reduce the maximum number of process that can be active.
- Init prevents the system from too many zombies.
- The ps(1) command prints Z to indicate the process status & a zombie process as zombie.
- Ex: #include<stdlib.h>

```
#include<sys/types.h>
#include<unistd.h>
int main()
{
    int pid;
    pid_t child_pid;
    child_pid = fork( );
    if (child_pid == 0)
    {
        exit(0);
    }
    else{
        sleep(3);
        system("pspid,ppid,status,cmd");
    }
}
```

- Syntax : #include<sys/type.h>
#include<unistd.h>
Pid_t fork(void);

2) Vfork()

- It creates a new process.
- The purpose of creating a new process is to execute a new program.
- The calling sequence & return values are similar to the fork() function.
- Function returns 0 in the child process & the new child's process ID in the parent.
- it does not copy the address space of the parent into child bcoz the child will not refer that address space.
- Syntax : #include<sys/type.h>
#include<unistd.h>
Pid_tvfork(void);

3) exit()

- it terminates a function normally.
- It takes an argument called status, that helps to examine the exit status of a process. if,
It is called without the exit status.
Main returns without a return value.
Main "falls off the end", then the exit status of the process is undefined.
- Syntax: #include<stdlib.h>
Void exit(int status);
- When it is called, it performs all the cleaning operations of the standard I/O library.
- Before closing the program, the fclose() function is called for all open streams.

4) wait()

- It waits for the first child to terminate.
- It blocks the caller until a child process terminates.
- it does not support job control
- It suspends execution of the current process until one of its children terminates
- Syntax: #include<sys/wait.h>

Pid_t wait(int *status_loc);

5) waitpid()

- It waits for the particular child to terminate.
- It can prevent the caller from blocking
- it support job control.
- It suspends execution of the current process until a child specified by pid argument has changed state .
- Syntax:
#include<sys/wait.h>

ii) **fg**

it will resume execution of the process by sending it a CONT signal.

SIGNAL GENERATION AND HANDLING

Signal generation

- Every signal has a symbolic name starting with SIG.
- The signal names are defined in signal.h, which must be included by any C program that uses signals.
- The names of the signals represent small integers greater than 0.
- 2 signals SIGUSR1 & SIGUSR2 are available for users & do not have a preassigned use.
- SIGFPE or SIGSEGV are generated when certain errors occur, other signals are generated by specific calls such as alarm.
- Generate signals from the shell with the kill command.

SIGNAL	DESCRIPTION	DEFAULT ACTION
SIGABRT	Process abort	Implementation dependent
SIGALRM	Alarm clock	Abnormal termination
SIGBUS	Access undefined part of memory object	Implementation dependent
SIGCHLD	Child terminated, stopped/continued	ignore
SIGCONT	Execution continued if stopped	continue
SIGFPE	Error in arithmetic operation as in division by zero	Implementation dependent
SIGHUP	Hang-up on controlling terminal	Abnormal termination
SIGIL	Invalid hardware instruction	Implementation dependent
SIGINT	Interactive attention signal	Abnormal termination
SIGKILL	Terminated	Abnormal termination
SIGPIPE	Write on a pipe with no readers	Abnormal termination
SIGQUIT	Interaction termination, core dump	Implementation dependent
SIGSEGV	Invalid memory reference	Implementation dependent
SIGSTOP	Execution stopped	stop
SIGTERM	Termination	Abnormal termination
SIGSTP	Terminal stop	stop
SIGTTIN	Background process attempting to read	stop
SIGTTOU	Background process attempting to write	stop
SIGURG	High bandwidth data available at a socket	ignore
SIGUSR1	User-defined signal1	Abnormal termination
SIGUSR2	User-defined signal2	Abnormal termination

Signal handling

- It takes two arguments i.e., signo & handler.
- The signo is the name of the signal like SIGINT defined in the <signal.h> header. The handler is a pointer to a user defined signal handler function.
- This function takes a signo as formal argument & returns void as the return type.
- On success, the signal function returns a pointer to the previous signal handler for a signal.
- It returns an error with a value SIG_ERR.
- The handler argument can take different values as given in table:

value	meaning
SIG_IGN	Ignore the signal. SIGKILL & SIGSTOP cannot be ignored
SIG_DFL	Perform default action
Address of signal handler	Call the signal handler

Advantages

- To perform some other action instead of default action.
- To block a signal temporarily.
- To restore the signal handler for a signal after it has been altered.

UNRELIABLE SIGNALS

- Unreliable signals means when a signal occurs, it could get lost, that is the process would never know the occurrence of the signals.
- Also, the process has control over the signals that is a process can only catch the signal or ignore it.
- But it can not block a signal so that it can retrieve the signal when it is ready.
- All signals were unreliable in earlier versions of linux.

Problem with unreliable signals

With earlier versions of linux is that each time the signal occurs, the action for this signal was reset to its default action.

Ex: the following snapcode shows the way the earlier systems handled the interrupt signal i.e., SIGINT.

```
intsigint_handler();
:
Signal(SIGINT, sigint_handler);
Sigint_handler( )
{
Signal(SIGINT, sigint_handler);
.....
}
```

The problem with this code is that, after the first occurrence of the SIGINT signal, if the interrupt signal occurs another time just before the call to signal function in the signal

int raise(intsig_no);

- Sig_no is the signal number of a signals to be sent to one or more processes indicated by pid.
- Pid is to kill the process ID of the recipient.
- Both kill & raise system calls returns 0 on success & -1 on error.

Alarm FUNCTION

- It is used by a process to set timer.
- The timer expires after a certain number of real clock seconds.
- When the timer expires, the kernel sends the SIGALRM signal to the calling process.
- The default action for the signal is to terminate the process.
- If there is a previously registered alarm clock for the process that has not yet expired then the call to alarm function returns the number of CPU sec left in process timer.
- The call to alarm cancels the effect of the previous alarm call & reset the timer with new alarm clock.
- Syntax: #include<unistd.h>

Unsigned int alarm(unsigned inttime_in_seconds);

Pause FUNCTION

- It suspends the execution of the calling process until a signal is caught.
- It will be aborted only when a signal is caught & that signal handler returns.
- The function returns -1 with errno set to EINTR.
- The alarm & pause functions can be used to implement the sleep function.
- Syntax : #include<unistd.h>

Int pause(void);

abort FUNCTION

- It terminates a process abnormally.
- It generates SIGABRT signal.
- When a process calls the abort function, it terminates the process & never returns to the caller.
- If the process does not terminate itself from the signal handling function, the abort function terminates the process when the signal handler returns.
- Syntax: #include<stdlib.h>

Void abort(void);

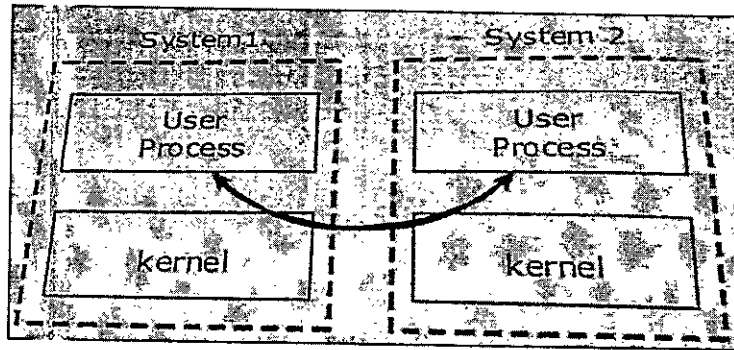
Sleep FUNCTION

- It suspends the caller for the specified number of CPU seconds.
- The calling process wakes up if either the time specified by secs argument is elapsed.
- It returns a value 0 if the specified number of sec has elapsed.

- In case the sleep returns early bcoz the process caught a signal, the sleep returns the number of sec it could not sleep.
- The actual return time of the sleep may be at a time later than requested, bcoz of the processor scheduling delays.

Syntax: `#include<unistd.h>`

`Unsigned int sleep(unsigned int secs);`



The above fig explains the IPC between two processes on different computer systems.

Pipes-creation:

Pipes

Linux system uses pipes to establish interprocess communication mechanism allowing two or more processes to send information to each other.

They are commonly used from within shells to connect the standard output of one utility to the standard input of another.

A pipe provides unidirectional flow of data. A pipe is created using the pipe() function

Syntax:

```
#include <unistd.h>
```

```
int pipe(intfd);
```

A pipe function returns two file descriptors fd[0] and fd[1]. The fd[0] is open for reading from and fd[1] is open for writing to the pipe. The data flows from one end of the pipe to the other end.

The pipe function returns '0' on success or '-1' on error.

IPC between related processes using unnamed pipes:

Unnamed pipes may be only used with related processes (parent/child or child/child having the same parent). They exist as long as their descriptors are open.

Syntax:

```
#include <unistd.h>
```

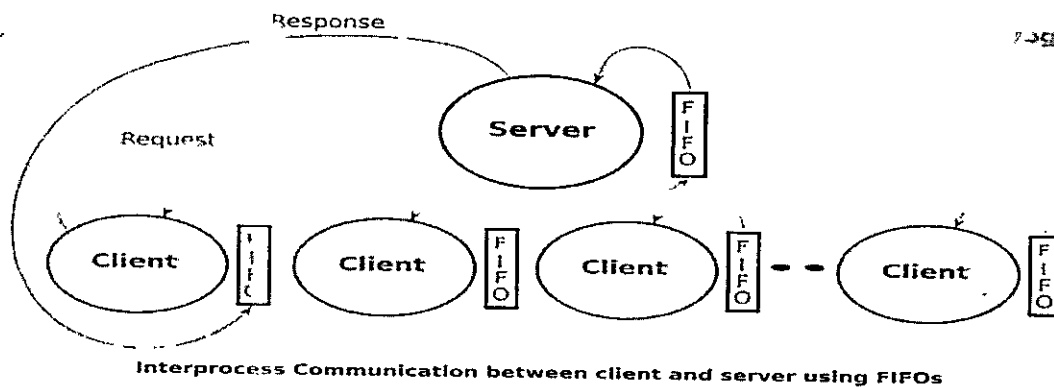
```
int pipe(intfd[2])
```

RETURNS: success : 0 error: -1

mkfifo creates a named pipe called fileName.

IPC between unrelated processes using FIFO:

The client-server paradigm comprises of a single server process, which works all the time, receives requests from clients and gives them responses. A client is the process that manages the inputs and outputs for a live user. Clients come and go but the server works all the time. The clients communicate with the server using an interprocess communication mechanism. Each process in the paradigm has a system-wide mechanism for receiving messages. In the example in a later section, we will use the FIFO as the mechanism for receiving messages. That is, the server will have a FIFO, where clients can put messages for the server. Similarly, each client will have a FIFO, where the server can put in messages for that client.



A FIFO can be opened using the open system call. After open, we can use the read and write system calls for reading from and writing to the FIFO, using the file descriptor returned by open. Of course, as per our design, we will either read or write to a FIFO but not do both. A process, be it a client or the server, reads from its own FIFO for receiving data and writes on other process's FIFO for sending data to that process.

Differences between Unnamed and Named Pipes:

Unnamed pipe:

- 1) These are created by the shell automatically.
- 2) They exist in the kernel.
- 3) They can not be accessed by any process, including the process that creates it.
- 4) They are opened at the time of creation only.

MESSAGE QUEUES:

API'S FOR MESSAGE QUEUES:

The `<sys/ipc.h>` header defines a structure `ipc_perm` data types that stores the owner and creator and group IDs. The assigned name key and the read – write permissions of a message queue.

The message table entry data type is `struct msqid_ds`, that is defined in the `<sys/message.h>`

The following mentioned are the different Unix system V API's for messages,

- 1) `msgget()`
- 2) `msgsnd()`
- 3) `msgrcv()`
- 4) `msgctl()`

1) `msgget()`

The `msgget` system call is used to get the message queues in Unix programming environment.

Syntax:

```
#include<sys/msg.h>
```

```
int msgget(key_t key, int msgflag);
```

The `msgget()` argument returns the message queue identifier associated with `key`.

2) `msgsnd()`

This system call is used for sending messages in an Unix environment.

Syntax:

```
#include<sys/msg.h>
```

```
int msgsnd(int msqid, const void *msgp, size_t msgsz, int msgflg);
```

The `msgsnd()` function is used to send a message to the queue associated with the message queue identifier specified by `msqid`.

The `msgp` argument points to a user-defined buffer that contain first a field of type `long int` that will specify the type message. and then a data portion that will hold the data bytes of the message.

The structure below is an example of what this user-defined buffer might look like, `struct mymsg`

```
{
```

The following are the unix API'S for semaphores,

- 1) Semget().
- 2) Sempot().
- 3) Semctl().

The function definitions of semaphores are,

```
#include<sys/types.h>
```

```
#include<sys/ipc.h>
```

```
#include<sys/sem.h>
```

```
Intsemctl(intsem_id, intsem_num, int command, ...);
```

```
Intsemget(key_t, intnum_sems, intsem_flags);
```

Intsemop(intsem_id, structsembuf, *sem_ops, size_tnum_sem_ops);

- 1) segment():

The semgnt() API function serves two fundamental roles. Its first use is in the creation of new semaphores.

The second use is identifying an existing semaphore.

Syntax:

```
Intsemget(key_t key intnum_sems, intsem_flags);
```

The key argument specifies a system-wide identifier that uniquely identifies this semaphores. The key must be nonzero or special symbol IPC_PRIVATE. The IPC_PRIVATE variable tells semget that no key is provided and so simply make one up. Since no key exists, there's no way for other processes to know about this semaphore. Therefore, it's a private semaphore for this particular process.

The developer can create a single semaphore with an nsems values of one or multiple semaphore. If we're using semget to get an existing semaphore, this value can simply be zero.

The num_sems parameter is the number of semaphore required, this is almost always 1.

- 2) semop():

The semop() API function provides the means to acquire and release a semaphore or semaphore array. The basic operations provided by semop are to decrement/increment a semaphore.

API'S FOR SHARED MEMORY:

The following are the Unix API's for shared memory,

- 1) Shmget()
- 2) Shmat()
- 3) Shmctl()
- 4) Shmdt()

The following are the Unix API's for shared memory,

- 1) shmget()

This system call is used for getting a shared memory identifier.

Syntax:

`#include<sys/types.h>`

`#include<sys/ipc.h>`

`#include<sys/shm.h>`

`Intshmget(key_t key, size_t size, intshmflg);`

The shmget() function returns the shared memory identifier associated with key.

A shared memory identifier and associated data structure and shared memory segment of at least size bytes are created for key if one of the following are true,

- i) The key argument is equal to IPC_PRIVATE.
- ii) The key argument does not already have a shared memory identifier associated with it, and (shmflg&IPC_CREAT) is true.

- 2) shmat()

Syntax:

`#include<sys/types.h>`

`#include<sys/shm.h>`

`Void *shmat(intshmidx, const void* shmaddr, intshmflg);`

Default

```
#include<sys/ipc.h>
```

```
#include<sys/shm.h>
```

```
Intshmdt(void* addr);
```

On execution, the above function returns a value '0' to indicate success, otherwise -1 to indicate error.

Example:

```
#include<iostream.h>
```

```
#include<stdio.h>
```

```
#include<stdlib.h>
```

```
#include<string.h>
```

```
#include<sys/stat.h>
```

```
#include<sys/ipc.h>
```

```
#include<sys/shm.h>
```

```
int main()
```

```
{
```

```
Int perms = S_IRWXU | S_IRWXU | S_IRWXO;
```

```
Intfd = shmget (100 , 1024, IPC_CREAT | perms);
```

```
If(fd == -1)
```

```
perror("shmget");
```

```
exit(1);
```

```
char* addr = (char*)shmat(fd, 0, 0);
```

```
if(addr==(char*)-1)
```

```
perror("shmat");
```

```
exit(1);
```

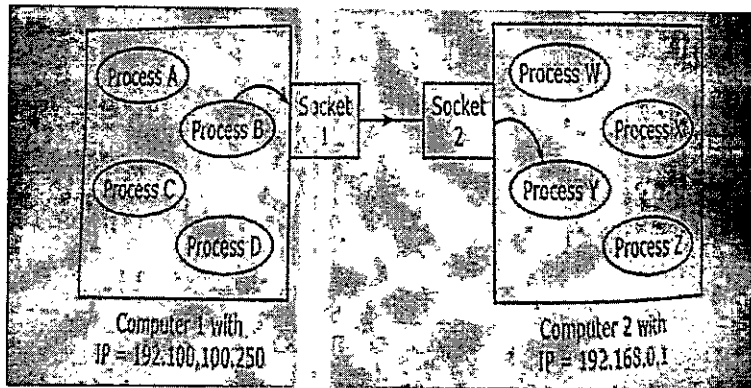
```
strcpy(addr, 'Hello');
```

```
if (shmdt(addr) == -1)
```

{protocol, foreign address, foreign process}

This half-association is referred to as the socket or transport address.

Two communicating processes maintain one socket each i.e., two half-associations, which makes one full association or connection.



SOCKET ADDRESS STRUCTURES:

A socket address structure is a special structure that stores the connection details of a socket. It mainly consists of fields like IP address, port number and protocol number and protocol family. Different protocol suites use different socket address structures. The different socket address structures are,

- 1) IPv4 socket address structure - "structure sockaddr_in".
- 2) IPv6 socket address structure - "structursockaddr_in6".
- 3) Generic socket address structure - "structure sockaddr".
- 4) New generic socket address structure - "socket _ storage".

IPv4 Socket Address Structure:

The IPv4 socket structure is also known as internet socket address structure. It is defined in header file <netinet/in.h> and is named as "sockaddr_in". The POSIX definition of sockaddr_in is shown in below.

Structin_addr

{

/*stores a 32-bit IPv4 address */

In_addr ts_addr;

/*specifies the address family of socket address structure for IPv6 it is AF_INET6*/

Sa_family_t sin6_port;

/*32-bit flow label. Its use is still under research*/

Unit32_t sin6_flowinfo;

/*128-bit IPv6 address as defined in struct in6_addr*/

Struct in6_addr sin6_addr;

/*32-bit scopeID*/

Unit32_t s_id;

};

Generic socket Address Structure

The functions present in socket API usually require an argument, which is a pointer to a socket address structure.

Any socket function that receives a parameter through pass-by-reference method does not come to know the type of the parameter. Then how can we declare the variable to receive that parameter.

The developers used the concept of generic socket address structure, which is defined in <sys/socket.h> header file and is shown below

Struct sockaddr

{

/*stores the length of structure it is usually, 8-bit unsigned integer*/

Unit8_t sa_len;

/*stores the address family. its value usually starts with AF_ followed by family name

Example: AF_INET, AF_INET6 etc*/

Sa_family_t sa_family;

/*Address belonging to specific protocols*/

Char sa_data[14]

SOCKET SYSTEM CALLS FOR CONNECTION ORIENTED PROTOCOL:

The system call listen is used by a connection-oriented server to get ready for accepting connection requests from a client.

Syntax:

```
#include<sys/types.h>
```

```
#include<sys/socket.h>
```

```
Intlisten(intsockfd, int backlog);
```

The first argument sockfd is a socket descriptor, as returned by a socket function call.

The second parameter backlog specifies the number of requests that can be queued by the system before the server executes the accept system call.

4) accept system call

This system call accept is used by connection-oriented server to set up an actual connection with a client process.

Syntax:

```
#include<sys/types.h>
```

```
#include<sys/socket.h>
```

```
Intaccept(intsockfd, structsockaddr *cli_addr, int *addrlen);
```

This system call returns a new socket descriptor.

5) connect system call

The system call connect is used by a client to establish a connection with the server.

Syntax;

```
#include<sys/types.h>
```

```
#include<sys/socket.h>
```

```
.. This connect(intsockfd, structsockaddr *servaddr, intaddrlen);
```

This system call is similar to accept. The server address pointed to by seraddr and its length addlen should be known.

6) write system call

It can be used as either send or write system call to regular messages.

`Intsocket(int family or domain, int type, int protocol);`

The first parameter family or domain specifies the communication protocol used.

The second parameter type specifies the type of socket.

2) bind system call

The system call bind associates an address to a socket descriptor created by socket. It binds a name to a socket.

Syntax:

```
#include<sys/types.h>
```

```
#include<sys/socket.h>
```

`Intbind(int sockfd, structsockaddr* myaddr, intaddrlen);`

The first parameter references the socket. The addrlen argument specifies the size of name structure pointed by the myaddr argument.

The second parameter myaddr specifies a pointer to a predefined address of the socket.

3) sendto system call

since datagram sockets aren't connected to a remote host, guess which piece of information we need to give before we send a packet.

Syntax:

```
#include<sys/types.h>
```

```
#include<sys/socket.h>
```

`Intsendto(int sockfd, const void *msg, intlen, unsigned int flags, conststructsockaddr *to, socklen_t tolen);`

4) recvfrom system call

This system call's function is same as the recv system call.

Syntax:

```
#include<sys/types.h>
```

```
#include<sys/socket.h>
```

`Intrecvfrom(int sockfd, void *buf, intlen, unsigned int flags, structsockaddr *from, int *fromlen);`

Teaching Plan

VIDYA JYOTHI INSTITUTE OF TECHNOLOGY

(AUTONOMOUS)

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

SUBJECT: LINUX PROGRAMMING

ACADEMIC YEAR: 2019-20

NAME: M.VIJAYA

YEAR/SEM/SECTION: III B.TECH/I SEM/A

Teaching Plan

Lecture No	TOPICS	Teaching Learning Process
UNIT I		
1	File handling utilities	Chalk& talk
2	Security by file permissions	Chalk& talk
3	Process utilities, Disk utilities,	Power Point Presentation
4	Networking commands, Filters,	Power Point Presentation
5	Text processing utilities Backup utilities,	Power Point Presentation
6	sed – scripts, operation, addresses, commands, applications	Power Point Presentation
7	Awk-execution, fields and records,	Power Point Presentation
8	scripts, operation, patterns, actions,	Power Point Presentation
9	functions, using system commands in awk	Chalk& talk
10	REVISION	Power Point Presentation
UNIT II		
11	Shell programming with Bourne again shell (bash): Introduction	Chalk& talk

12	shell responsibilities, pipes and input Redirection, output redirection	Roll Play
3	here documents, running a shell script, the shell as a programming language	Chalk& talk
4	shell meta characters	Chalk& talk
5	file name substitution, shell variables,	Chalk& talk
6	command substitution substitution, shell variables, command substitution	Chalk& talk
7	shell commands, the environment, quoting.	Chalk& talk
8	test command, control structures,	Chalk& talk
9	arithmetic in shell, shell script examples	Chalk& talk
10	interrupt processing. functions.	Chalk& talk
11	debugging shell scripts	Chalk& talk
12	shell scripts Examples	Power Point Presentation
13	shell scripts Examples	Power Point Presentation
14	REVISION	Chalk& talk

UNIT III

1	Files - File Concept, File types,	Chalk& talk
2	File System Structure, file metadata-Inodes,	Chalk& talk
3	kernel support for files, system calls for file I/O operations- open, create, read, write, close, lseek, dup2,	Chalk& talk
4	file status information-stat family, file and record locking- fcntl function, file permissions - chmod, fchmod,	Chalk& talk
5	file ownership-chown, lchown. links-soft and hard links - symlink, link, unlink.	Chalk& talk

6.	Directories -Creating, removing and	Chalk& talk
7	changing Directories-mkdir, rmdir, chdir,	Chalk& talk
8	obtaining current working directory-getcwd, Directory contents	Chalk& talk
9	Scanning Directories-opendir, readdir,	Chalk& talk
10	closedir, rewinddir functions.	Chalk& talk

UNIT IV

1	Process - Process concept,	Chalk& talk
2	Kernel support for process, process identification,	Chalk& talk
3	process control - process creation, replacing a process image,	Chalk& talk
4	waiting for a process,	Power Point Presentation
5	process termination,	Power Point Presentation
6	zombie process,	Power Point Presentation
7	orphan process.	Power Point Presentation
8	Signals - Introduction to signals, Signal generation and handling	Chalk& talk , Think-Pair-Share
9	Kernel support for signals. Signal function.	Chalk& talk
10	unreliable signals, reliable signals,	Chalk& talk
11	kill, raise, alarm	Chalk& talk
12	pause, abort, sleep functions.	Chalk& talk

UNIT V

1	Inter Process Communication - Introduction to IPC	Chalk& talk
2	IPC between processes on a single computer system,	Chalk& talk

3	IPC between processes on different systems.	Chalk& talk
4	pipes-creation.	Chalk& talk
5	IPC between related processes	Chalk& talk
6	using unnamed pipes, FIFOs- creation,	Chalk& talk
7	IPC between unrelated processes using FIFOs(Named pipes).	Chalk& talk
8	differences between unnamed and named pipes,	Chalk& talk
9	popen and pclose library functions.	Chalk& talk
10	Message Queues APIs for message queues	Chalk& talk
11	Semaphores, APIs for semaphores	Chalk& talk
12	Shared Memory APIs for shared memory	Chalk& talk
13	Sockets, Socket address structures	Power Point Presentation
14	Socket system calls for connection oriented protocol	Power Point Presentation
15	connectionless protocol	Power Point Presentation
16	Revision	Chalk& talk

Total No of Classes:62


Course Coordinator


CSE-HOD

Assignment Questions

VIDYA JYOTHI INSTITUTE OF TECHNOLOGY

(Accredited by NAAC & Approved by A.I.C.T.E., New Delhi, Permanently Affiliated to JNTU, Hyderabad)

**(Aziz Nagar, C.B.Post, Hyderabad- 500075)
(AUTONOMOUS)**

COMPUTER SCIENCE & ENGINEERING

ASSIGNMENT I

Branch: CSE

Year&Sem: III-I

SUB: LP

Academic Year: 2019-20

Faculty Name: M.VIJAYA

Marks: 25M

Sno	Question Number	Marks	CO	BL	PO's
1	Explain in detail about Linux Operating system structure.	5	1	L2	1-5,7,8,10-12
2	Explain about file access permission.	5	1	L2	1-5,7,8,10-12
3	Define Shell? Responsibilities of Shell?	5	2	L1	1-5,8-12
4	Explain about Control statements.	5	2	L2	1-5,8-12
5	Explain about standard i/o system calls.	5	3	L2	1-5,8,10-12

VIDYA JYOTHI INSTITUTE OF TECHNOLOGY

(Accredited by NAAC & Approved by A.I.C.T.E., New Delhi. Permanently Affiliated to JNTU.
Hyderabad)

(Aziz Nagar, C.B.Post, Hyderabad- 500075)
(AUTONOMOUS)

COMPUTER SCIENCE & ENGINEERING

ASSIGNMENT II

Branch: CSE

Year&Sem: III-I

SUB: LP

Academic Year: 2019-20

Faculty Name: M.VIJAYA

Marks: 25M

Sno	Question Number	Marks	CO	BL	PO's
1	What is file attributes.	5	3	L1	1-5,8,10-12
2	Explain the following system calls with syntax:(i) lseek() (ii) read() (iii)open () (iv) creat()	5	4	L2	1-5,8-12
3	What is fork system call?	5	4	L1	1-5,8-12
4	What are IPC Between processes on a single user system?	5	5	L1	1-8,11,12
5	What is semaphore. Also explain the APIs associated for semaphore	5	5	L1	1-8,11,12

Mid I & Mid II
Question Papers



Vidya Jyothi Institute of Technology (Autonomous)

(Accredited by NAAC & NBA, Approved By A.I.C.T.E., New Delhi, Permanently Affiliated to JNTU, Hyderabad)
(Aziz Nagar, C.B.Post, Hyderabad -500075)

III Year B.Tech 1st Semester 1st Mid Examination

Branch: CSE	Duration: 90Min
Sub: LINUX PROGRAMMING	Marks: 20
Date: 13.8.2019	Session: AN

Course Outcomes:

1. Understand and make effective use of linux utilities.(LEVEL 2)
2. Able to write shell scripts to solve the problems.(LEVEL 2)
3. Develop the skills necessary for file system and directory handling.(LEVEL 5)
4. Learn the concepts of process and signal system calls(LEVEL 2)
5. Implement inter process on mechanisms.(LEVEL 5)

Bloom Levels:

Remember	I
Understand	II
Apply	III
Analyze	IV
Evaluate	V
Create	VI

PART-A (3Q×2M =6Marks)		Outcomes		B.L	M
ANSWER ALL THE QUESTIONS		CO	PO		
1	Write the syntax with example for the following commands a)cat b)sort c)chmod	CO1	1-5,11,12	V	2
2	Define shell? List different types of shell?	CO2	1-5,8-12	I	2
3	What is file? List types of files?	CO3	1-5,8,10-12	I	2
PART-B (5+5+4= 14 Marks)		Outcomes		B.L	M
ANSWER ALL THE QUESTIONS		CO	PO		
4.i)	Explain the File Handling utilities?	CO1	1-5,11,12	II	5
[OR]					
ii)	Explain about AWK command with example?	CO1	1-5,11,12	II	5
5.i)	Explain about responsibilities of shell?	CO2	1-5,8-12	II	5
[OR]					
ii)	Explain control structures with example?	CO2	1-5,8-12	II	5
6.i)	Draw and explain Linux File System Structure?	CO3	1-5,8,10-12	II	4
[OR]					
ii)	Explain the following system calls with syntax? a) fopen() b) fclose() c) fseek() d) fflush()	CO3	1-5,8,10-12	II	4



Vidya Jyothi Institute of Technology (Autonomous)

(Accredited by NAAC & NBA, Approved By A.I.C.T.E., New Delhi, Permanently Affiliated to JNTU, Hyderabad)
(Aziz Nagar, C.B.Post, Hyderabad -500075)

III Year B.Tech 1st Semester 2nd Mid Examination

Branch: CSE

Duration: 90Min

Sub: LINUX PROGRAMMING

Marks: 20

Date:29.10.2019

Session: FN

Course Outcomes:

1. Understand and make effective use of linux utilities.(LEVEL 2)
2. Able to write shell scripts to solve the problems.(LEVEL 2)
3. Develop the skills necessary for file system and directory handling.(LEVEL 5)
4. Learn the concepts of process and signal system calls(LEVEL 2)
5. Implement inter process ion mechanisms.(LEVEL 5)

Bloom Levels:

Remember	I
Understand	II
Apply	III
Analyze	IV
Evaluate	V
Create	VI

PART-A (3Q×2M =6Marks)		Outcomes		BL	Marks
ANSWER ALL THE QUESTIONS		CO	PO		
1.i)	Explain the following system calls with syntax. (a)chdir() (b) closedir()	CO3	1-5,8,10-12	II	2
[OR]					
ii)	What is hard link?	CO3	1-5,8,10-12	II	2
2.i)	Explain the following system calls for signals a) kill() b) raise()	CO4	1-5,8-12	II	2
[OR]					
ii)	What is signal function?	CO4	1-5,8-12	II	2
3.i)	What is pipe? How to create a pipe?	CO5	1-8,10-12	II	2
[OR]					
ii)	What is socket? How to create a socket?	CO5	1-8,10-12	II	2
PART-B (4+5+5= 14 Marks)		Outcomes		BL	Marks
ANSWER ALL THE QUESTIONS		CO	PO		
4.i)	What is file? Explain kernel support for files?	CO3	1-5,8,10-12	II	4
[OR]					
ii)	Write a C program to implement cp command by using system calls?	CO3	1-5,8,10-12	I	4

5.i)	Define zombie process. Write a program to illustrate the zombie process concept.	CO4	1-5,8-12	I	5
[OR]					
ii)	What is signal? Differentiate between reliable and unreliable signals	CO4	1-5,8-12	II	5
6.i)	Describe various APIs of Message queues that are used for inter process communication.	CO5	1-8,10-12	I	5
[OR]					
ii)	Describe Socket system calls used for connectionless protocol with syntax and usage.	CO5	1-8,10-12	II	5

VJIT(A)

Unit Wise Questions

LINUX PROGRAMMING

Unit : I

Long Answers Type Questions : (5 questions)

1. a) Explain in detail about Linux Operating system structure.
b) Explain file handling utilities?
2. Write in detail about five Text Processing Utilities.
3. Explain about process utilities?
4. a) Explain various patterns and actions in awk.
b) Write an awk script to perform simple arithmetic operations
5. Explain the commands in sed.

Short Answers Type Questions : (20 qsns)

1. What is Linux?
2. What is kernel and explain its functions?
3. Explain security by file permissions?
4. Explain any three process utilities?
5. Write the syntax with example for the following commands
a) chmod b) tr c) tar
6. Explain about any three filters?
7. Explain disk utilities?
8. Write differences between sed and awk?
9. Explain the networking commands?
10. Explain about SED Addresses with example?
11. Explain about AWK command and patterns in AWK?
12. Write the syntax with example for the following commands
a) grep b) sort c) telnet
13. Explain about head and tail command?
14. Write short notes on SED command?
15. Explain about comparing commands? (comm, diff, cmp)
16. Explain backup utilities (*tar*, *cpio*)?
17. Differentiate between a process, a program and a job?
18. Write a short note on buffers in AWK?
19. What is the difference between append and insert command in SED?
20. Define filter?

Unit : II

Long Answers Type Questions : (5 questions)

1. Define Shell? Responsibilities of Shell?
2. Write about the types of shells? Explain the shell commands?
3. write about control statements with syntaxes?
4. Write a shell script to count the specified number of lines in a text file without using wc command?
5. a) With an example script explain the differences between 'while' and 'until' statements.
b) List and explain the various meta characters available in shell programming.

Short Answers Type Questions : (20 qsns)

1. What is shell?
2. Types of shells?
3. Write short notes on I/O redirection operators?

4. Define the here document with example?
5. Write about Responsibilities of Shell?
6. Write a shell script for arithmetic operations using case statement?
7. Write a shell script to find the reverse of the number?
8. Describe about various shell variables?
9. What is Test Command?
10. Write a shell script to find the factorial of a given number?
11. Define function? Write a list of predefined functions?
12. Write about the types of shells? and Meta characters in shell?
13. Write a shell script to find file or directory?
14. Describe about control statements with syntaxes?
15. Explain various Meta characters in shell with an example script?
16. Write a shell script to illustrate cat command in Linux?
17. Explain how debugging can be done in a shell script?
18. What is command substitution?
19. Write a shell script to find and delete all file with the word "Unix"?
20. Write a short notes on interrupt processing ?

Unit : III

Long Answers Type Questions : (5 questions)

1. Explain about file system structure in Linux.
2. Explain the following system calls with syntax:
(i) lseek() (ii) read() (iii) open() (iv) creat()
3. Explain about hard and symbolic links with examples
4. Discuss the data structures that support the linux files in detail?
5. a) Explain about scanning directories functions.
b) Write a c program to implement ls command by using system calls?

Short Answers Type Questions : (20 qsns)

1. List the standard I/O functions?
2. Define file descriptor.
3. What is i-node?
3. Difference between stream pointer and file descriptor?
4. What is system call?
5. List the scanning directories functions?
6. List the system calls for Directories ?
7. What is Symbolic link?
8. What is hard link?
9. Define file? Write the types of files?
10. Write a program to implement cp command using system call?
11. Write a program to implement mv command using system calls?
12. What are file attributes?
13. write a program to implement cat command using system calls?
14. What is the command used for changing the directory?
15. what is meant by reference counter?
16. Explain the following system calls with syntax;
(a) mkdir() (b) rmdir()
17. Explain the following system calls with syntax. (a) chdir() (b) closedir()
18. What is links with examples?

19. Explain in detail about various files.
20. Write a program to explain link system call

Unit : IV

Long Answers Type Questions : (5 questions)

1. What is meant by Process? Explain the following with example:
 - (a) Process Creation
 - (b) Process Termination
2. What is an orphan process? Write a program to illustrate orphan process.
3. What is an Zombie process? Write a program to illustrate Zombie process.
4. a) Difference between fork() and vfork()
b) Difference between reliable and unreliable signals
5. Explain the below system calls with the help of syntax and examples:
a) kill b) raise c) alarm d) pause e) abort

Short Answers Type Questions : (20 qsns)

1. What is meant by Process?
2. What is Process Creation?
3. What is Process Termination?
4. Differentiate between real IDs and effective IDs?
5. What is an orphan process?
6. What is an Zombie process?
7. Differentiate between thread and process?
8. Explain the following system calls for signals
a) kill() b) raise()
9. Explain the following system calls for signals
c) alarm() d) abort()
10. Explain about the kernel support for signals.
11. What is signal handler? explain with an example.
12. What are the signals that are not ignored or blocked?
13. What is need of exec() system call? Write syntax?
14. Differentiate between fork() and vfork().
15. Explain about the kernel support for processes.
16. What is signal function?
17. What are process identifiers? Mention the commands for getting different IDs of calling process.
18. Write a program that demonstrates the use of exit().
19. difference between wait() and waitpid()?
20. difference between signal and interrupt?

Unit : V

Long Answers Type Questions : (5 questions)

1. Define unnamed pipe? How do we create unnamed pipe? Explain the limitations of unnamed pipe.
2. Define named pipe? How do we create named pipe? Write c programs that illustrate communication between two unrelated processes using named pipe?
3. Describe various APIs of Shared memory that are used for inter process communication.
4. Describe various APIs of Message queues that are used for inter process communication.
5. a) Explain briefly about the following socket APIs with clear syntax:
i) socket() ii) bind() iii) listen()

b) Describe Socket system calls used for connectionless protocol with syntax and usage.

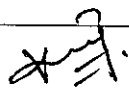

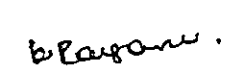
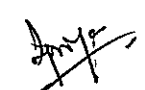
Short Answers Type Questions : (20 qsns)

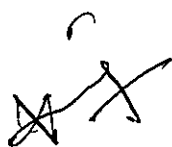
1. What is IPC?
2. What are IPC Between processes on a single user system?
3. What is socket? How to create a socket?
4. What is pipe? How to create a pipe?
5. What is shared memory?
6. What is FIFO explain with example?
7. Compare the IPC mechanisms?
8. Explain with example the Kernel Support for message queues?
9. How to create message queues?
10. What is msgsend and msgrcv system calls?
11. List the API for Shared memory?
12. Explain with example the Kernel Support for semaphore?
13. Difference between connection oriented and connection less protocols?
14. Write the syntax for semop(), semget(), semctl() system calls?
15. How to control semaphore?
16. What is semaphore? Types of semaphore?
17. what are the connection less socket methods ?
18. What is generic socket address structure?
19. Differentiate between pipe() and FIFO.
20. What is IPv4 and IPv6 socket address structure?

Minutes of Course Review Meeting

Meeting 1

Date: 8/7/2019

Details of Meeting No – 1	
Date of Meeting	8/7/2019
Member's Present	1. M.VIJAYA 2. K.SAMATHA 3. Dr.K. RANGA RAO 4. P.LAKSHMI PRIYA
Details	Points discussed in the meeting: <ul style="list-style-type: none">• Preparation of Unit wise questions and give assignment to students• Teaching Learning Practices• Status of Syllabus coverage of Mid I.
Signatures	1.  2.  3.  4. 



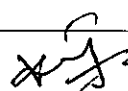

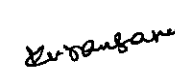
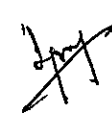
Course Coordinator



CSE-HOD

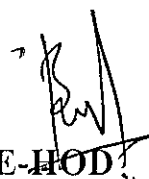
Meeting 2

Date: 14/10/2019

Details of Meeting No – 2	
Date of Meeting	14/10/2019
Member's Present	1. M.VIJAYA 2. K.SAMATHA 3. Dr.K. RANGA RAO 4. P.LAKSHMI PRIYA
Details	Discussion on <ul style="list-style-type: none">• Preparation of unit wise questions and giving assignment• Status of Syllabus coverage
Signatures	1.  2.  3.  4. 



Course Coordinator



CSE-HOD

Lecture Notes

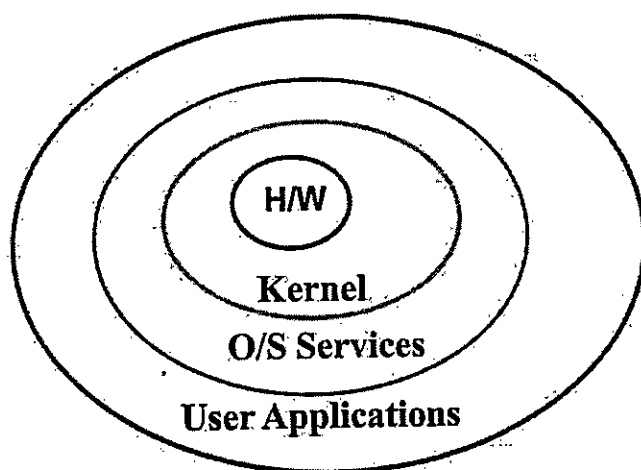
UNIT-1

Linux is a free and open source software operating system that can perform all the tasks that any current popular O.S can. The defining component of a Linux distribution is its kernel, which is the heart of an operating system. It controls the hardware, CPU, memory, hard disk, network card etc.

The shell acts as an interface which enables the communication between user and kernel. It interprets the inputs of the user as commands and passes them to the kernel.

Linux-kernel architecture:

High level architecture:



Kernel : It is the heart of the OS. It is the main program in the UNIX System. It controls Hardware, Software, CPU, memory, hard disk, network card etc.

Shell : It is the interface between user and the kernel. It interprets your commands and pass them to kernel.

Application : Provides useful functions for the OS.

Welcome

(Ctrl-d)-to save the content entered after the command.

To display the content -

\$cat file1

2. rm -

In Unix file system, to remove files we use this command.

Syntax: \$rm. [-option] <filename>

Options:

-f : (force) forcibly removes the file though it has all r, w, x permissions.

-r : (recursive removal) removes all files and empty directories. Even if the directory has files, it removes the files first and then removes the directory.

-i : (interactive removal) remove command with interactive flag asks the user before removing the file or directory.

Example: \$rm -f file1

File naming and renaming

1. move -

Syntax: \$mv [-option] [file/dir] [newfile/newdir]

Options:

-i : (interactive flag) The interactive flag with mv command is useful to warn the user that the destination file already exists.

-f : (force) this is used to forcibly overwrite the already existed file.

Example: \$mv -i file1 file2

Editing files

In Unix file system, we edit a file, append or update the existing data by using vi editor.

Syntax: \$vi <filename>

Example: \$vi file1

Esc + i --- insert mode(to enter the contents)

Esc : wq --- to save the contents

File access permissions

Unix operating system supports 3 kinds of permissions.

1.read 2.write 3.execute

read : read permissions allows to read the content of a file

write : write permissions allows to edit / move/ delete a file.

execute : execute permissions allows to execute the program files and shell script files for different sets of data.

2.who –

This command is used to know who is currently logged on to the system.

Syntax : \$who [-option]

Options :

-H : displays the header of the column.

-U : provides a detailed list.

Example: \$who -H

3.w –

w command is same as the who command except that this gives detailed output of user activities and many details of the system.

Syntax: \$w [-options]

Options:

-H, -W

4.kill –

It kills the process id.

Syntax: \$kill signal PID PID

5.pkill –

It kills the process.

Syntax: \$kill signal process process

Disk utilities

1. du-

It displays the disk space used by the specified file/directory. And it displays disk usage of the current directory.

Syntax: \$du [-option]

Options:

-a: (all) count of all files, not just for directory.

-b: (byte) it prints the size in bytes.

-t: (total) it prints the grand total.

-h: (human readable) it prints the size in human readable format.

2. df-

This command is used to find out the empty disk space.

Syntax: \$df

5. **tr**: (translating characters) used for translating letters from character set to other.

Syntax: **\$tr [-option] string1 string2**

This command replaces the characters specified in string1 with characters specified in string2. These strings are enclosed in double quotes. The translation is done by replacing the first character in string1 with first character in string2.

Options:

- d: this option is used to delete from text line that are matching during translation
- s: if same characters are occurring, this is used to compress the char in the output.
- c: this option is used to display lowercase letters to uppercase.

Examples:

\$tr -d "legtwy"

Yesterday I dared to struggle today I dare to win

Srda I dard o strugg oda I dar o in

\$tr -s "ria" "egr"

Yesterday I dared to struggle today I dare to win

Yesteedry I dreed to steuggle todry I dreed to wgn.

\$tr -c "abc" "ABC"

Stu aokabc ode

Stu Aok ABC ode

Filters

In Unix, a filter is any command that has the following features.

- 1) receives its input from standard input
- 2) manipulates input
- 3) sends the result to the standard output

Examples:

more, cat, comm, cut, cmp, diff, head, tail, paste, sort, tr, uniq, wc.

1. **more** -

It is a command to view the contents of a text file one screen at a time.

Syntax: **\$more [-option] <filename>**

2. **sort** -

It sorts the contents of a file i.e. arranges the content in a particular sequence using ASCII character code values.

It is also responsible for merging and comparison.

Syntax: **\$sort [-option] <filename>**

Options:

- c: It checks the given file that is already sorted.
- n: It sorts on numeric values.

Example:

7. uniq -

This command displays unique lines in a file. It displays only single copy of each line on to the standard output.

Syntax: `$uniq [-option] <filename>`

Options:

-u: this option compresses the output of duplicated lines and display the only unique lines in a file.

-d: it is a complement of -u option. It displays single copy of duplicated lines.

-c: this option counts the occurrence of each line.

Comparing commands

There are 3 commands that are used to compare the contents of two files.

1)cmp 2)comm 3)diff

1. cmp -

This command is used to perform comparison of 2 files byte by byte. Its used to determine whether the file is identical/not.

Syntax: `$cmp [-option] filename1 filename2`

Options:

-l: this option displays the list of all differences present in a file byte by byte.

-s: this is known as suppress list option. This doesn't display any output.

2. comm -

This command prints the lines that are common in 2 files as input.

Syntax: `$comm [-option] filename1 filename2`

3.diff -

This command prints the difference between 2 files by performing comparison line by line and by comparing first file with second file. When a difference is marked, the first file is altered inorder to match it with the second file.

Syntax: `$diff [-option] filename1 filename2`

Networking commands

The commands which we will use for communication between more than one system for sharing resources or any other issue are known as networking commands.

These are of 5 types.

1)telnet 2)finger 3)ftp 4)arp 5)rlogin

1. telnet -

2. tar -

This is used to save and restore the files and directories to/from different types of media like tape or floppy disk. We can mount a directory or format a disk to which we want to backup data without creating a file system on it.

Syntax: `$tar -cvf tarfile/tapedevice directory`

sed command

This is the most powerful filter. It stands for stream editor. A stream editor is used to perform basic text transformations on an input stream (a file or input from a pipeline). While in some ways similar to an editor which permits scripted edits (such as `ed`), `sed` works by making only one pass over the input(s), and is consequently more efficient. But it is `sed`'s ability to filter text in a pipeline which particularly distinguishes it from other types of editors.

It reads standard input process using a file called `sed` script and writes the result to the standard output.

Syntax: `$sed [-option] 'address.action' file(s)`

Options:

- e: It is a default option. It indicates that the script is on the command line.
- f: It indicates that the script is in a file which immediately follows this option.
- n: It suppresses the automatic output i.e. it will not display the contents of pattern space.

Example: `$sed -f myscript.sed f.txt`

sed script

A `sed` script is a file that contains list of instructions to be applied to each line in an input file. If there is only one instruction, it can be included as command line.

If there are more instructions, that need to be executed frequently, they should be saved in a separate file. Each instruction in a `sed` script consists of an address and a command separated by a complement operator.

sed addresses

A `sed` address is an instruction which determines the lines in the input file that are to be processed or skipped by the command in the instructions.

Under this there are four:

1. single address
2. set of line addresses
3. range address
4. nested address

single address:

awk is a more utility. The operations of awk is similar to sed utility. It reads a standard input line by line and takes an action on a part of entire line. The actions are specified in an awk script that consists of list of instructions.

Each instruction contains a pattern and its associated action.

Syntax: `awk [-option] awk_script file(s)`

awk execution

awk script is a file containing a list of instructions to be applied to each line in a input file.

Syntax: `$awk 'pattern{action}' filename`

Example: `$awk '/program/{print}' f1.txt`

Fields and records

A file is viewed as collection of fields and records by the awk utility. A field is a data unit that gives some data. Each line in a file is a record, which is a collection of several fields. However the record contains related data.

A file organized into records is called a data file.

Scripts

These are divided into 3 parts:

1. initialization
2. body
3. end of job

BEGIN {action}
pattern1 {action1}
pattern2 {action2}
.
patternN {actionN}
END {action}

1. Initialization –
This part defines instruction for initializing variables , creating report headings , sed system variables etc. It is identified with a token BEGIN and all the instructions are enclosed with curly braces.
This part is processed only once before the awk reads the first line from the input file.
2. Body –
It consists of one or more instructions for processing the data in a file. Each instruction consists of a pattern associated with an action that will be taken when the pattern is matched.

3. `substring()` –
The `substring()` returns substring from the given string.
Syntax: `substr(string , starting_index)` or
`substr(string , starting_index , length)`
4. `split()` –
This divides the string into 2 substrings using a field separator.
Syntax: `split (str ,array)`
`split (str ,array ,field_separator)`
5. `sub()` –
The `sub()` substitutes one string for another string that matches your regular expression.
It returns 1 if the string was substituted and returns 0 if it failed.
Syntax: `sub (regular_exp , with_string , input_string)`
6. `gsub()` –
This is same as `sub()` except that `gsub()` substitutes all the occurrences of the matching string with another string.
Syntax: `gsub(regular_exp ,with_string , input_string)`

4) Bourne Again Shell (BASH):-

The popularity of sh motivated programmers to develop a shell that was compatible with it, but with several enhancements. Linux systems still offer the sh shell, but "bash" – the "Bourne-again Shell," based on sh – has become the new default standard. One attractive feature of bash is its ability to run sh shell scripts unchanged. Shell scripts are complex sets of commands that automate programming and maintenance chores; being able to reuse these scripts saves programmers time. Conveniences not present with the original Bourne shell include command completion and a command history.

5) The T C Shell

Tcsh fixed problems in csh and added command completion, in which the shell makes educated "guesses" as you type, based on your system's directory structure and files. Tcsh does not run bash scripts, as the two have substantial differences.

Shell script:-

The usage of more number of commands in one file is called Shell script.

*Shell responsibilities

When the shell encounters a meta character "*", then it replaces the list of files from the current directory that matches the pattern.

Command substitution:-

The shell executes the command surrounded by a backward quotes.

Syntax:- `command`

Eg:- \$echo enter the date is `date`

Sequences:-

A shell executes a series of commands in a sequence from left to right and the sequence of commands are separated by a semicolon.

Eg:- \$cat > file1 ; ls ; pwd

Background processing:-

When a user gives a command or a series of commands followed by a "&" symbol and a "&metacharacter", a subshell is created to execute the commands in a background which will run simultaneously as parent shell and act as the background process.

Eg:- \$fact.c & date&

Subshells:-

A shell consists of a parent shell and a child shell. A current shell creates a new shell to perform a specific task.

Shell has two data areas

Environmental area

Local variable area

Variables:-

There are two types of shell variables:-

Local variables

Environmental variables

The data in these variables is stored as string.

A child shell inherits a copy of the parent shell which is called as the environmental variables but not local variable.

The useful information transmitted by using environmental variable.

Grouping commands:-

Shell allows group of commands separated by using semicolon by placing between the parenthesis.

The group of commands is executed by using sub shell or child shell.

Eg:- \$(who;pwd;date;ls)

*Pipes:-

Piping is a process of combining 2 or more commands using a pipe operator("|").

Eg:- \$who | ls

The output from command on the left will be the input to the other command.

Quotes

There are mainly four types of quotes characters used in Linux . These are

1. " (double quote):

The double quote ("quote") protects everything enclosed between two double quote marks except \$. ' , " and \. Use the double quotes when you want only **variables and command substitution**.

2. ' (single quote):

The single quote ('quote') protects everything enclosed between two single quote marks. It is used to **turn off the special meaning** of all characters.

3. ` (back quote):

Backtick is not a quotation sign. it has a very special meaning. Everything you type between backticks is evaluated (executed) by the shell before the main command (like shown in your examples), and the *output* of that execution is used by that command, just as if you'd type that output at that place in the command line.

4. \ (back slash):

The backslash (\) alters the special meaning of the ' and " i.e. it will escape or cancel the special meaning of the next character.

Control Structures

The *control flow* commands alter the order of execution of commands within a shell script. They include the **if...then, for...in, while, until**, and **case** statements. In addition, the **break** and **continue** statements work in conjunction with the control flow structures to alter the order of execution of commands within a script.

1. If.. then.. fi

The if statement tests the result of a command and then conditionally executes a group of statements. It is a nested conditional flow control.

Programs

1. Write a shell script using two arguments.

Ans.

```
$ vi file1
```

```
Echo "My First number is $1"
```

```
Echo "My Second number is $2"
```

```
Echo "total number of arguments are $#"
```



Power Point Presentation

INDEX

S No. Topics

1. Working with Bourne shell
2. Shell responsibilities
3. Pipes, redirection
4. Here documents
5. Shell meta characters
6. Shell variables
7. Shell commands
8. Environment
9. control structures
10. Shell script examples

2

What's Shell?

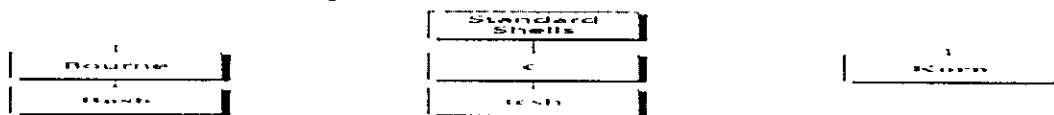
It's acts an interface between the user and OS (kernel). It's known as "command interpreter".

When you type ls :

shell finds cmd (/usr/bin).

shell runs cmd.

you receive the output



Shell responsibilities

1. Program Execution
2. Variable and Filename Substitution
3. I/O Redirection
4. Pipeline Hookup
5. Environment Control
6. Interpreted Programming Language

11

Shell metacharacters

The shell consists of large no. of metacharacters. These characters play a vital role in Unix programming.

Types of metacharacters:

1. File substitution
2. I/O redirection
3. Process execution
4. Quoting metacharacters
5. Positional parameters
6. Special characters

28

Environment Variables

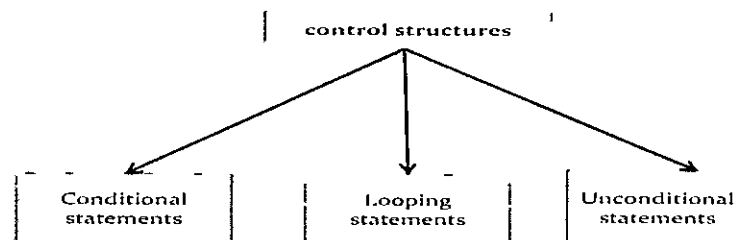
- They are initialized when the shell script starts and normally capitalized to distinguish them from user-defined variables in scripts
- To display all variables in the local shell and their values, type the **set** command
- The **unset** command removes the variable from the current shell and sub shell

Environment Variables

Environment Variables	Description
\$HOME	Home directory
\$PATH	List of directories to search for commands
\$PS1	Command prompt
\$PS2	Secondary prompt
\$SHELL	Current login shell
\$0	Name of the shell script
\$#	No. of parameters passed
\$\$	Process ID of the shell script

38

control structures



File Concepts

- > Files are the building blocks of any OS.
- > A file can be defined as an entity by which information is stored in a UNIX file system.
- > The commands we use, the applications we use, the data we store and the devices we access are all contained in a file.
- > A file can also be defined as a container which is used to store the information.
- > In the UNIX operating system, the kernel itself is treated as a file.

File Types

There are various types of files on a UNIX OS. They are:-

1. Regular Files
2. Directory File
3. Device file
 - a. Character Device File
 - b. Block Device File
4. FIFO file
5. Socket file
6. Symbolic link file

"Everything is a File" and Types of Files in Linux

Normal	-	Normal file
Directory	d	Normal directory
Hard link	-	Additional name for existing file
Symbolic link	l	Shortcut to a file or directory
Socket	s	Pass data between 2 process
Named pipe	p	Like sockets, user can't work directly with it
Character device	c	Processes character in communication
Block device	b	Major and minor numbers for controlling dev

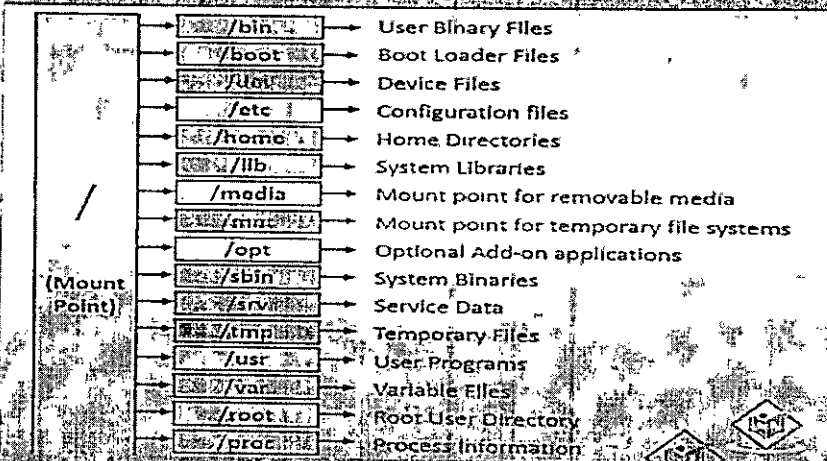
Linux File Systems

- > In Unix each and every hardware component or device treated as file.
- > All file in Unix are related to one another.
- > The file system in Unix is a collection of all these related files organized in a hierarchical structure.
- > Root directory = "/"
- > Absolute path name - starts from "/"
- > Relative path name - may start with "." or ".."

Linux File System Structure

Linux File system Structure defines the directory structure and directory contents. Unlike operating systems, it is maintained by the Linux Foundation.

Linux Directory Structure



socket

- **Socket** - create an endpoint for communication

SYNOPSIS

```
#include <sys/types.h> /* See NOTES */
#include <sys/socket.h>
int socket(int domain, int type, int protocol);
```

DESCRIPTION

- **socket()** creates an endpoint for communication and returns a file descriptor that refers to that endpoint. The file descriptor returned by a successful call will be the lowest-numbered file descriptor not currently open for the process.

Semester End Question Papers



Vidya Jyothi Institute of Technology (Autonomous)

(Accredited by NAAC & NBA, Approved By A.I.C.T.E., New Delhi, Permanently Affiliated to JNTU, Hyderabad)
(Aziz Nagar, C.B.Post, Hyderabad -500075)

Subject code: A15512.

III B. Tech I SEM REGULAR EXAMINATION NOVEMBER 2017

LINUX PROGRAMMING

(COMMON TO CSE & IT)

Time: 3hrs

Max.Marks:75

Note: This question paper contains two PARTS, PART A and PART B.
PART A is compulsory which carries 25 marks. Answer all questions.
PART B consists of 5 Units. Answer any one full question from each unit

PART - A

ANSWER ALL THE QUESTIONS

25 M

1. What are the advantages of Linux operating system? 2M
2. Explain which character is used to search a pattern in the beginning of each line using grep command. 3M
3. List shell responsibilities in Linux. 2M
4. Describe about various shell variables. 3M
5. Define file descriptor. 2M
6. Explain the meaning of . and .. with respect to directory. 3M
7. Define Zombie Process. 2M
8. Differentiate between thread and process. 3M
9. Define semaphore. 2M
10. Draw the structure of message queue for storing 3 messages in message queues. 3M

PART-B

ANSWER ALL THE QUESTIONS

5X10M=50M

- 11.i) (a) Create a file with name employee which stores ename, epid, designation, salary. Write a command to display the details ename, epid, and salary whose designation is "Manager".

- (b) Discuss file attributes? Explain how to change basic file permissions with examples.

[OR]

- ii) Write short notes on the following utilities:

(A) cat (B) cp (C) wc (D) rm

- 12.i) (a) Write about the types of shells? Explain the shell commands.

- (b) Write a shell script to print the Fibonacci series below 50 which is given as:

0 1 1 2 3 5 8 13 21 34.

[OR]

- ii) (a) Explain Responsibilities of Shell.

- (b) Explain the structure of if-else and case statements of a shell program with suitable example.

13. i) Write about File and Directory maintenance system calls? (Give syntax & examples).

[OR]

- ii) (a) Write a C-program for 'wc' command using system calls or library functions.

- (b) Write short notes on Inode.

- 14.i) (a) Explain the Kernel support for Process with a neat diagram.?

- (b) What is an orphan process? Write a program to illustrate orphan process?

[OR]

- ii) (a) Differentiate between wait() and waitpid().

- (b) Explain the function of SIGCHLD and SIGQUIT signals.

15. i) a) Define FIFOs? How they are different from pipes? Give an example application where FIFO can be used.

- b) Explain IPv4 and IPv6 socket address structure.

[OR]

- ii) (a) Explain similarities and dissimilarities between the semaphore and shared memory-IPC Mechanisms.

- (b) What is socket? Explain various socket system calls used for TCP protocol.

VJIT (A)



Vidya Jyothi Institute of Technology (Autonomous)

(Accredited by NAAC & NBA, Approved By A.I.C.T.E., New Delhi, Permanently Affiliated to JNTU, Hyderabad)

(Aziz Nagar, C.B.Post, Hyderabad -500075)

Subject code: A15512

III B. Tech I SEM REGULAR EXAMINATION - NOVEMBER 2018

LINUX PROGRAMMING (COMMON TO CSE &IT)

Time: 3hrs

Max.Marks:75

Note: This question paper contains two PARTS A and B.

PART A is compulsory which carries 25 marks. Answer all questions.

PART B carries 5 questions. Answer all the questions.

PART - A

ANSWER ALL THE QUESTIONS

25 M

1. List text processing Linux utilities. 2M
2. What is the purpose of sed. Give an example? 3M
3. What is the significance of here documents? 2M
4. Write a shell script to find the reverse of a given number. 3M
5. What is rewinddir function? 2M
6. What is File? Explain various file attributes. 3M
7. What is fork() function? Give an example. 2M
8. What is the use of alarm function? 3M
9. What is FIFO? Why FIFO's are called as named pipes? 2M
10. Explain about shmctl () function. 3M

PART-B

ANSWER ALL THE QUESTIONS

5X10M=50M

- 11.i) a) Discuss backup utilities with examples?
b) What is the purpose of awk? Explain with examples.

OR

- ii) a) Write an awk script that reads a file of which each line has 5 fields – ID, NAME, MARKS1, MARKS2, MARKS3 and finds out the average for each student. Print out the average marks with appropriate messages.
b) Explain various file handling utilities with suitable examples.

- 12.i) a) List and explain various quoting? Explain with examples.
b) Write a shell script to find reverse of a given number?

OR

- ii) a) Explain test command with suitable example?
b) List and explain interrupt processing functions.

13. i) a) Explain in detail about the file I/O operations.
b) Differentiate system calls and library functions.

OR

- ii) a) Explain the file system structure in Linux system.
b) Give syntaxes of commands mkdir, rmdir, chdir and explain.

14. i) a) Explain the signal concept with an example.
b) What is a zombie process? Explain with an example.

OR

- ii) a) What is an orphan process? Write a program to illustrate orphan process?
b) What are reliable and unreliable signals? Explain.

15. i) a) Compare the IPC provided by shared memory and message queue.
b) Explain bind(), listen(), accept() methods of TCP socket.

OR

- ii) a) What are pipes? Explain how pipes are created and used in IPC with examples.
b) What is Socket? Explain socket system calls for connection oriented protocol.

VJIT(A)

Extra Topics Delivered

Extra Topics Delivered:

S.No	Name of the Topic
1	Communication utilities
3	Socket Programming

1. The ping Utility The ping command sends an echo request to a host available on the network. Using this command, you can check if your remote host is responding well or not. The ping command is useful for the following – Tracking and isolating hardware and software problems. Determining the status of the network and various foreign hosts. Testing, measuring, and managing networks. Syntax Following is the simple

syntax to use the ping command – \$ping hostname or ip-address

2. Server side C/C++ program to demonstrate Socket programming

```
#include <unistd.h>
#include <stdio.h>
#include <sys/socket.h>
#include <stdlib.h>
#include <netinet/in.h>
#include <string.h>
#define PORT 8080
int main(int argc, char const *argv[])
{
    int server_fd, new_socket, valread;
    struct sockaddr_in address;
    int opt = 1;
    int addrlen = sizeof(address);
    char buffer[1024] = {0};
    char *hello = "Hello from server";

    // Creating socket file descriptor
    if ((server_fd = socket(AF_INET, SOCK_STREAM, 0)) == 0)
    {
        perror("socket failed");
        exit(EXIT_FAILURE);
    }

    // Forcefully attaching socket to the port 8080
    if (setsockopt(server_fd, SOL_SOCKET, SO_REUSEADDR | SO_REUSEPORT,
                   &opt, sizeof(opt)))
    {
        perror("setsockopt");
    }
}
```



```

        exit(EXIT_FAILURE);
    }
    address.sin_family = AF_INET;
    address.sin_addr.s_addr = INADDR_ANY;
    address.sin_port = htons( PORT );

    // Forcefully attaching socket to the port 8080
    if (bind(server_fd, (struct sockaddr *)&address,
              sizeof(address))<0)
    {
        perror("bind failed");
        exit(EXIT_FAILURE);
    }
    if (listen(server_fd, 3) < 0)
    {
        perror("listen");
        exit(EXIT_FAILURE);
    }
    if ((new_socket = accept(server_fd, (struct sockaddr *)&address,
                             (socklen_t *)&addrlen))<0)
    {
        perror("accept");
        exit(EXIT_FAILURE);
    }
    valread = read( new_socket , buffer, 1024);
    printf("%s\n",buffer );
    send(new_socket , hello , strlen(hello) , 0 );
    printf("Hello message sent\n");
    return 0;
}

```

Client side C/C++ program to demonstrate Socket programming

```

#include <stdio.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <string.h>
#define PORT 8080

int main(int argc, char const *argv[])
{
    int sock = 0, valread;
    struct sockaddr_in serv_addr;
    char *hello = "Hello from client";
    char buffer[1024] = {0};
    if ((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0)
    {
        printf("\n Socket creation error \n");
        return -1;
    }

    serv_addr.sin_family = AF_INET;
    serv_addr.sin_port = htons(PORT);

```

```

// Convert IPv4 and IPv6 addresses from text to binary form
if(inet_pton(AF_INET, "127.0.0.1", &serv_addr.sin_addr)<=0)
{
    printf("\nInvalid address/ Address not supported \n");
    return -1;
}

if (connect(sock, (struct sockaddr *)&serv_addr, sizeof(serv_addr)) < 0)
{
    printf("\nConnection Failed \n");
    return -1;
}
send(sock , hello , strlen(hello) , 0 );
printf("Hello message sent\n");
valread = read( sock , buffer, 1024);
printf("%s\n",buffer );
return 0;
}

```

Innovation in Teaching and Learning



Vidya Jyothi Institute of Technology

(Accredited by NAAC & NBA , Approved By A.I.C.T.E., New Delhi) permanently affiliated JNTUH

(Aziz Nagar, C.B.Post, Hyderabad -500075)

(AUTONOMOUS)

Innovative /Student Centric Teaching Method Form

Innovation in Teaching Learning: Role Play

Subject: Linux Programming

Name of the Faculty: M.VIJAYA

Topic: Shell Responsibilities

Class/ Section: III B.Tech I-Sem CSE-A

Teaching is an art and science. Teaching is a process of imparting knowledge and skills. It is a systematic process based on some educational objectives to communicate.

Interactive learning is a hands-on, real-world approach to education. 'Interactive learning actively engages the students in wrestling with the material. It reinvigorates the classroom for both students and faculty. Lectures are changed into discussions, and students and teachers become partners in the journey of knowledge acquisition.'

Role-playing is the changing of one's behaviour to assume a role, either unconsciously to fill a social role, or consciously to act out an adopted role.

- To refer to the playing of roles generally such as in a theatre, or educational setting;
- To refer to taking a role of an existing character or person and acting it out with a partner taking someone else's role, often involving different genres of practice

Shell responsibilities

Built in commands:-

A shell contains several present commands upon giving a command \$ls.

Wild cards or File name substitution:-

A wild card is also known as the file name substitution. A shell offers a wild card facility that helps in selecting files from a file system, that satisfies the specific pattern. Eg:- \$ls -l f-name*

Command substitution:-

The shell executes the command surrounded by a backward quotes. Syntax:- `command`

Sequences:-

A shell executes a series of commands in a sequence from left to right and the sequence of commands are separated by a semicolon. Eg:- \$cat > file1 ; ls ; pwd

Background processing:-

When a user gives a command or a series of commands followed by a "&" symbol and a "&metacharacter", a subshell is created to execute the commands in a background which will run simultaneously as parent shell and act as the background process. Eg:- \$fact.c & date&

Subshells:-

A shell consists of a parent shell and a child shell. A current shell creates a new shell to perform a specific task.

Variables:-

There are two types of shell variables:- Local variables, Environmental variables

Grouping commands:-

Shell allows group of commands separated by using semicolon by placing between the parenthesis.

Eg:- \$(who;pwd;date;ls)

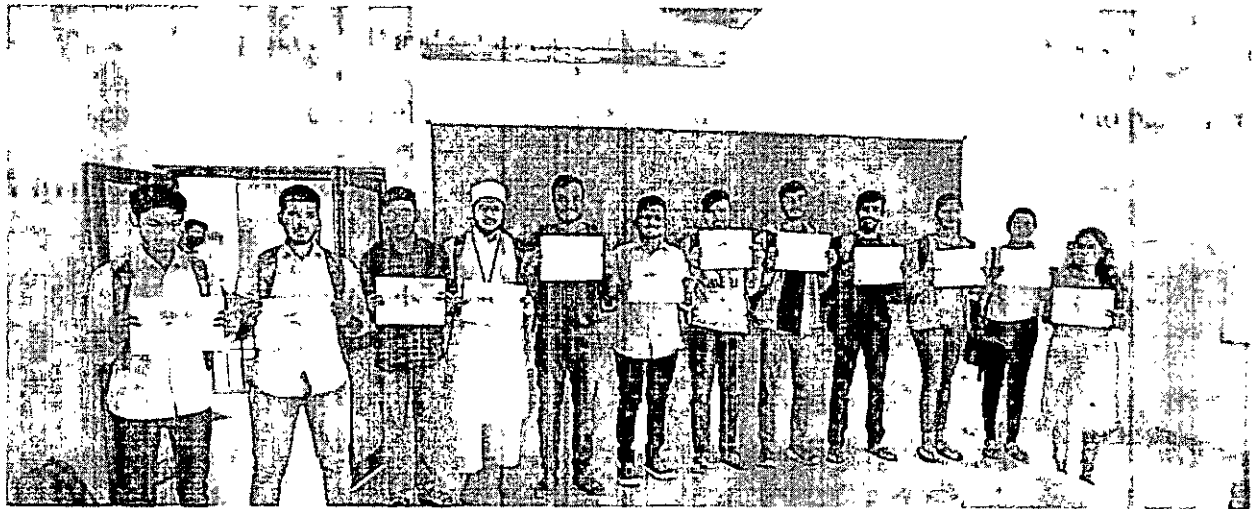
Pipes:-

Piping is a process of combining 2 or more commands using a pipe operator("|").

Eg:- \$who | ls

Redirection:-

Redirection is a process in which we use a file in place of one of the standard streams.

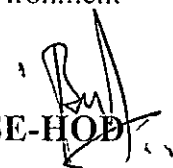


OUTCOME:

- Develops communication and language skills
- Develops social skills as students collaborate with others
- Encourages students to express their ideas and feelings in a relaxed environment


Instructor

M.Vijaya


CSE-HOD



Vidya Jyothi Institute of Technology

(Accredited by NAAC & NBA , Approved By A.I.C.T.E., New Delhi) permanently affiliated JNTUH

(Aziz Nagar, C.B.Post, Hyderabad -500075)

(AUTONOMOUS)

Innovation in Teaching Learning: Think-Pair-Share

Subject: LINUX PROGRAMMING

Name of the Faculty: M.VIJAYA

Topic: Signals

Class/ Section: III B.Tech I-Sem CSE-A

Think-Pair-Share (TPS) is a collaborative learning strategy in which students work together to solve a problem or answer a question about an assigned reading. This technique requires students to (1) think individually about a topic or answer to a question; and (2) share ideas with classmates. Discussing an answer with a partner serves to maximize participation, focus attention and engage students in comprehending the reading material.

Benefits:

- The Think-Pair-Share strategy is a versatile and simple technique for improving students' reading comprehension.
- It gives students time to think about an answer and activates prior knowledge.
- TPS enhances students' oral communication skills as they discuss their ideas with one another.
- This strategy helps students become active participants in learning and can include writing as a way of organizing thoughts generated from discussions.

T : (Think) Teachers begin by asking a specific question about the text. Students "think" about what they know or have learned about the topic.

P : (Pair) Each student should be paired with another student or a small group.

S : (Share) Students share their thinking with their partner. Teachers expand the "share" into a whole-class discussion.

Execution:

A. T: Faculty asked question about "Signals". Students think individually for 5 minutes.

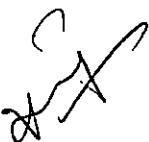
B. P: Each student paired with another student as a small group and should think among themselves or within a group for 5 minutes.

Then They have written the points what they discussed. Both have them actively participated and shared the necessary points on topic.

C. S: As they shared their thinking with their partner they came up with Signal concept . They shared their work into a whole class discussion.



Outcome: The Think-Pair-Share activity gives them the opportunity to feel more comfortable sharing their thoughts.


Instructor
M. Vijaya


CSE-HOD

**Assessment Sheet – Co Wise
(Direct Attainment)**

Faculty:

S.No	Reg.No	MID-I Threshold 60%					MID-II Threshold 60%					Threshold 60%				
		ASM-I (S)	Q1(2M)	Q2(2M)	Q3 B (2M)	Q4(5M)	PART-B Q5(5M)	Q6(4M)	ASM-II (S)	Q1(2M)	Q2(2M)		Q3 A (2M)	Q4(4M)	PART-B Q5(5M)	Q6(5M)
1	17601A0511	5	2	2	2	4	4	4	5	2	2	1	4	4	4	53
2	17601A0575	5	2	2	2	2	2	3	5	2	2	1	3	2	2	51
3	17911A0501	5	2	2	2	2	2	2	5	2	2	1	2	2	2	50
4	17911A0502	5	2	2	0	2	2	2	5	2	2	2	1	2	2	59
5	17911A0503	5	2	2	2	2	2	4	5	2	2	2	1	4	2	46
6	17911A0504	5	2	2	2	3	3	3	5	2	2	2	3	3	3	63
7	17911A0505	5	2	2	2	3	3	2	5	2	2	2	2	3	3	49
8	17911A0506	5	2	2	2	4	4	4	5	2	2	2	4	4	4	54
9	17911A0507	5	2	2	2	5	5	4	5	1	2	2	4	5	5	49
10	17911A0508	5	2	2	2	3	3	3	5	1	2	2	1	3	3	50
11	17911A0509	5	2	2	2	1	3	3	5	1	2	2	3	1	3	59
12	17911A0510	5	2	2	2	3	3	4	5	2	2	2	4	3	3	55
13	17911A0511	5	2	2	2	4	2	4	5	2	2	2	4	4	5	60
14	17911A0512	5	2	2	2	5	3	4	5	2	2	2	4	5	5	45
15	17911A0513	5	2	2	2	3	3	3	5	2	2	2	2	4	5	55
16	17911A0514	5	2	2	2	1	3	3	5	2	2	2	3	1	3	56
17	17911A0515	5	2	2	2	3	3	3	4	5	2	2	4	3	3	59
18	17911A0516	5	2	2	2	4	5	4	5	2	2	2	4	4	5	57
19	17911A0517	5	2	2	2	3	3	3	5	1	2	2	3	3	3	45
20	17911A0518	5	2	2	2	3	3	2	5	1	2	2	2	3	3	53
21	17911A0520	5	2	2	2	4	4	4	5	2	2	2	4	4	4	56
22	17911A0521	5	2	2	2	3	3	3	5	2	2	2	3	3	3	51
23	17911A0523	5	2	2	2	4	4	4	5	2	2	2	3	3	3	46
24	17911A0524	5	2	2	2	2	2	2	5	2	2	1	3	2	2	49
25	17911A0526	5	2	2	2	2	2	2	5	2	2	2	2	2	2	45
26	17911A0527	5	2	2	0	2	2	2	5	2	2	1	2	2	2	19
27	17911A0528	5	2	2	2	2	4	4	5	2	2	2	4	2	2	48
28	17911A0529	5	2	2	2	3	3	3	5	2	2	2	3	3	3	46
29	17911A0530	5	2	2	2	3	3	2	5	2	2	2	2	3	3	49
30	17911A0531	5	2	2	2	4	4	4	5	2	2	2	4	4	4	47
31	17911A0562	5	2	2	2	5	5	4	5	1	2	2	4	5	5	18
62	17911A0558	5	2	2	2	3	3	3	5	1	2	2	1	3	3	45
58	17911A0534	5	2	2	2	1	3	3	5	1	2	2	3	1	3	49
34	17911A0535	5	2	2	2	3	3	3	4	5	2	2	4	3	3	52
35	17911A0536	5	2	2	2	4	2	4	5	2	2	2	4	4	5	57
36	17911A0537	5	2	2	2	5	3	4	5	2	2	2	4	5	5	48
37	17911A0538	5	2	2	2	3	3	3	5	2	2	2	4	5	5	45
38	17911A0539	5	2	2	2	1	3	3	5	2	2	2	1	3	3	12
39	17911A0540	5	2	2	2	3	3	4	5	2	2	2	3	1	3	58
40	17911A0541	5	2	2	2	4	5	4	5	2	2	2	4	3	3	52
41	17911A0542	5	2	2	2	3	3	3	5	1	2	2	4	4	5	58
42	17911A0543	5	2	2	2	3	3	2	5	1	2	2	3	3	3	54
43	17911A0544	5	2	2	2	4	4	4	5	2	2	2	4	4	4	49
44	17911A0545	5	2	2	2	3	3	3	5	2	2	2	3	3	3	16
45	17911A0546	5	2	2	2	4	4	4	5	2	2	2	1	4	4	58
46	17911A0547	5	2	2	2	2	2	3	5	2	2	2	3	2	2	44
47	17911A0548	5	2	2	2	2	2	2	5	2	2	1	3	2	2	49
48	17911A0549	5	2	2	2	0	2	4	5	2	2	2	2	2	2	57
49	17911A0550	5	2	2	2	2	4	4	5	2	2	1	4	2	2	50
50	17911A0551	5	2	2	2	3	3	3	5	2	2	2	3	3	3	49
51	17911A0552	5	2	2	2	3	3	2	5	2	2	2	2	3	3	49
52	17911A0553	5	2	2	2	4	4	4	5	2	2	2	4	4	4	58



53	17911A0554	5	2	2	2	2	5	5	4	5	1	2	2	2	4	5	5	45
54	17911A0555	5	2	2	2	2	5	3	3	5	1	2	2	2	1	3	3	54
55	17911A0556	5	2	2	2	2	1	3	3	5	1	2	2	2	3	1	3	59
56	17911A0557	5	2	2	2	2	3	3	4	5	2	2	2	2	4	3	3	54
57	17911A0558	5	2	2	2	2	4	2	4	5	2	2	2	2	4	4	5	49
58	17911A0559	5	2	2	2	2	5	3	4	5	2	2	2	2	4	5	5	46
59	17911A0560	5	2	2	2	2	3	3	3	5	2	2	2	2	1	3	3	57
60	17911A0561	5	2	2	2	2	1	3	3	5	2	2	2	2	3	1	3	54
61	17911A0562	5	2	2	2	2	3	3	4	5	2	2	2	2	4	3	3	52
62	17911A0563	5	2	2	2	2	4	5	4	5	2	2	2	2	4	4	5	59
63	17911A0564	5	2	2	2	2	3	3	3	5	1	2	2	2	3	3	3	11
64	17911A0565	5	2	2	2	2	3	3	2	5	1	2	2	2	2	3	3	56
65	17911A0566	5	2	2	2	2	4	4	4	5	2	2	2	2	4	4	4	66
66	17911A0567	5	2	2	2	2	3	3	3	5	2	2	2	2	3	3	3	45
67	17911A0568	5	2	2	2	2	4	4	4	5	2	2	2	2	1	4	4	51
68	17911A0569	5	2	2	2	2	2	2	3	5	2	2	2	2	3	2	2	59
69	17911A0570	5	2	2	2	2	2	2	2	5	2	2	2	2	2	2	2	12
70	17911A0571	5	2	2	2	2	0	2	2	5	2	2	2	2	1	2	2	52
71	17911A0572	5	2	2	2	2	2	2	4	5	2	2	2	2	1	4	2	45
72	17911A0573	5	2	2	2	2	3	3	3	5	2	2	2	2	3	3	3	50
73	17911A0574	5	2	2	2	2	3	3	2	5	2	2	2	2	2	3	3	18
74	17911A0575	5	2	2	2	2	4	4	4	5	2	2	2	2	4	4	4	48
75	17911A0576	5	2	2	2	2	5	5	4	5	1	2	2	2	2	4	5	4
76	17911A0577	5	2	2	2	2	3	3	3	5	1	2	2	2	1	3	3	49
77	17911A0578	5	2	2	2	2	1	3	3	5	1	2	2	2	3	1	3	53
78	17911A0579	5	2	2	2	2	3	3	4	5	2	2	2	2	4	3	3	45
79	17911A0580	5	2	2	2	2	4	4	2	5	2	2	2	2	4	4	5	50
80	17911A0581	5	2	2	2	2	5	3	4	5	2	2	2	2	4	5	5	49
81	17911A0582	5	2	2	2	2	3	3	3	5	2	2	2	2	1	3	3	46
82	17911A0583	5	2	2	2	2	1	3	3	5	2	2	2	2	3	1	3	54
83	17911A0584	5	2	2	2	2	3	3	4	5	2	2	2	2	4	3	3	50
84	17911A0585	5	2	2	2	2	4	4	5	5	2	2	2	2	4	4	5	57
85	17911A0586	5	2	2	2	2	3	3	3	5	1	2	2	2	3	3	3	48
86	17911A0587	5	2	2	2	2	3	3	2	5	1	2	2	2	2	3	3	59
87	17911A0588	5	2	2	2	2	4	4	4	5	2	2	2	2	4	4	4	54
88	17911A0589	5	2	2	2	2	3	3	3	5	2	2	2	2	3	3	3	9
89	17911A0590	5	2	2	2	2	4	4	4	5	2	2	2	2	4	4	4	14
90	17911A0591	5	2	2	2	2	2	2	2	5	2	2	2	2	1	3	2	12
91	17911A0592	5	2	2	2	2	2	2	2	5	2	2	2	2	2	2	2	11
92	17911A0593	5	2	2	2	2	0	2	2	5	2	2	2	2	1	2	2	7
93	17911A0594	5	2	2	2	2	2	2	2	5	2	2	2	2	2	2	2	53
94	17911A0595	5	2	2	2	2	2	2	4	4	5	2	2	2	1	4	2	4
95	17911A0596	5	2	2	2	2	3	3	3	5	2	2	2	2	3	3	3	59
96	17911A0597	5	2	2	2	2	3	3	3	5	2	2	2	2	2	3	3	49
97	17911A0598	5	2	2	2	2	2	2	4	4	5	2	2	2	4	4	4	52
98	17911A0599	5	2	2	2	2	5	5	4	5	2	2	2	2	4	5	5	51
99	17911A0600	5	2	2	2	2	3	3	3	5	1	2	2	2	4	5	5	51
100	17911A0601	5	2	2	2	2	3	3	3	5	2	2	2	2	3	3	3	49
101	17911A0602	5	2	2	2	2	1	3	3	5	1	2	2	2	3	1	3	49
102	17911A0603	5	2	2	2	2	3	3	4	5	2	2	2	2	4	3	3	47
103	17911A0604	5	2	2	2	2	5	5	4	5	2	2	2	2	4	5	5	52
104	17911A0605	5	2	2	2	2	3	3	3	5	2	2	2	2	1	3	3	49
105	17911A0606	5	2	2	2	2	1	3	3	5	2	2	2	2	3	1	3	16
106	17911A0607	5	2	2	2	2	3	3	4	5	2	2	2	2	4	3	3	46
107	17911A0608	5	2	2	2	2	3	3	4	5	2	2	2	2	4	4	5	26
108	17911A0609	5	2	2	2	2	4	5	5	1	2	2	2	2	3	3	3	26
109	17911A0610	5	2	2	2	2	3	3	3	5	1	2	2	2	2	3	3	34
110	17911A0611	5	2	2	2	2	4	4	4	5	2	2	2	2	4	4	4	35
111	17911A0612	5	2	2	2	2	3	3	3	5	2	2	2	2	3	3	3	32
112	17911A0613	5	2	2	2	2	4	4	4	5	2	2	2	2	1	4	4	26
113	17911A0614	5	2	2	2	2	2	2	3	5	2	2	2	2	3	2	2	28
114	17911A0615	5	2	2	2	2	2	2	2	5	2	2	2	2	1	2	2	16
115	17911A0616	5	2	2	2	2	2	2	2	5	2	2	2	2	1	2	2	30

[illegible]



[illegible]

ASSESSMENT OF COs FOR THE COURSE							
			value	Avg	CO Attainment	CO Attainment [End	Overall CO Attainment
CO1	Method	ASM I	3	3.0	3.0	2.00	2.25
		MID I- PART A	3.0				
		MID I- PART B	3.0				
		ASM I- Q4	3				
CO2		ASM I- Q2	3.0	3.0	3.0	2.00	2.25
		MID I- PART A	3.0				
		MID I- PART B	3.0				
		ASM I- Q5	3				
CO3		ASM II	3	3.0	3.0	2.00	2.25
		MID I- PART A	3.0				
		MID I- PART B	3.0				
		ASM II- Q6	3.0				
CO4		ASM II- Q7	3.0	3.0	3.0	2.00	2.25
		MID II- PART A	3.0				
		MID II- PART B	3.0				
		ASM II- Q8	3				
CO5		ASM II- Q9	3.0	3.0	3.0	2.00	2.25
		MID II- PART A	3.0				
		MID II- PART B	3.0				
		ASM II- Q10	3				

Course End Survey Form



VIDYA JYOTHI INSTITUTE OF TECHNOLOGY

(Accredited by NBA, Approved by AICTE New Delhi & Permanently Affiliated to JNTUH)
Aziz Nagar Gate, C.B. Post, Hyderabad-500 075.

Department of Computer Science & Engineering

Course End Survey Form Academic year: 2019-20

Name of the student	Kochi Sandeep	Year & sem	III - I
Roll number	17911A0528	Regulations	R 15

Dear Student,

We need your help in evaluating the courses offered, by responding the short survey below.

Your feedback is very valuable for us in order to continually improve our program. The aim of this survey is to evaluate how well each of the courses has prepared you to have necessary skills.

Your responses will be kept confidential and will not be revealed to anyone outside the department without your permission.

Please indicate (✓) the level to which you agree with the following criterion:
(3: Strongly agree 2: Agree 1: Strongly disagree)

Name of The Course: LINUX PROGRAMMING		RATING		
After completing this course the student must demonstrate the knowledge and ability to		3	2	1
CO 1	Understand and make effective use of Linux utilities.	3		
CO 2	Able to write shell scripts to solve the problems.	3		
CO 3	Develop the skills necessary for file system and directory handling.	3		
CO 4	Learn the concepts of process and signal system calls.	3		
CO 5	Implement inter process communication mechanisms.	3		

Any other comments / suggestions: _____


Signature