Operating System Lab Manual
Course Code: CSE 323

Department of Computer Science and Engineering

SESSION 1: INTRODUCE

Intended Learning Outcome:

Installing Linux OS

To study about the basics of UNIX

Expected skills:

a. Gathering knowledge about installation.

Tools Required:

a. Ubuntu Operating System

Session Detail:

UNIX:

It is a multi-user operating system. Developed at AT & T Bell Industries, USA in 1969. Ken Thomson along with Dennis Ritchie developed it from MULTICS (Multiplexed Information and Computing Service) OS. By1980, UNIX had been completely rewritten using C language.

LINUX: It is similar to UNIX, which is created by Linus Torualds. All UNIX commands works in Linux. Linux is an open source software. The main feature of Linux is coexisting with other OS such as windows and UNIX.

STRUCTURE OF A LINUXSYSTEM:

It consists of three parts.

a) UNIX kernel

b) Shells

c) Tools and Applications

UNIX KERNEL: Kernel is the core of the UNIX OS. It controls all tasks, schedule all Processes and carries out all the functions of OS. Decides when one programs tops and another starts.

SHELL: Shell is the command interpreter in the UNIX OS. It accepts command from the user and analyses and interprets them.

Post Lab Exercise:

1. Discussion about basic of Ubuntu OS
2. Preparing Ubuntu operating system.

SESSION 2: INTRODUCE shell command

Intended Learning Outcome:

To study of Basic shell Commands

Expected skills:

a. Ubuntu windows ready for command,

Tools Required:

a. Ubuntu Operating System

Session Detail:

Any command that the shell internally executes is referred to as a shell command. There isn't an executable application that corresponds. Consider a CD as an example. Say that there is no /bin/cd program and that cd indicates that the command is a built-in command. It belongs to the shell. The shell processes the cd command (tcsh or bash, say).Compare that to ls. As shown by which ls, there is a program called /bin/ls. Ls is not internally processed by the shell.

sudo command

Short for superuser do, sudo is one of the most popular basic Linux commands that lets you perform tasks that require administrative or root permissions.

When using sudo, the system will prompt users to authenticate themselves with a password. Then, the Linux system will log a timestamp as a tracker. By default, every root user can run sudo commands for 15 minutes/session.

If you try to run sudo in the command line without authenticating yourself, the system will log the activity as a security event. Here's the general syntax:

Sudo You can also add an option, such as:

-k or –reset-timestamp invalidates the timestamp file.

-g or –group=group runs commands as a specified group name or ID.

-h or –host=host runs commands on the host.

pwd command

Use the pwd command to find the path of your current working directory. Simply entering pwd will return the full current path – a path of all the directories that starts with a forward slash (/). For example, /home/username.The pwd command uses the following syntax:

pwd [option]

It has two acceptable options:

-L or –logical prints environment variable content, including symbolic links.

-P or –physical prints the actual path of the current directory.

whoami

$whoami shows the currently logged-in user

ls comand

–used to list the files. Your files are kept in a directory.

Syn:

$ls List all files of that directory

ls–s All files (include files with prefix)

ls–l Lodetai (provide file statistics)

ls–t Order by creation time

ls– u Sort by access time (or show when last accessed together with –l)

ls–s Order by size

ls–r Reverse order

ls–f Mark directories with /,executable with* , symbolic links with @, local sockets with =, named pipes(FIFOs) with

ls–s Show file size

ls– h " Human Readable", show file size in Kilo Bytes & Mega Bytes (h can be used together with –l or)

ls[a-m]* List all the files whose name begin with alphabets From a to m

ls[a]* List all the files whose name begins with „a" or „A"

cd command

To navigate through the Linux files and directories, use the cd command. Depending on your current working directory, it requires either the full path or the directory name.

Running this command without an option will take you to the home folder. Keep in mind that only users with sudo privileges can execute it.

Let's say you're in /home/username/Documents and want to go to Photos, a subdirectory of Documents. To do so, enter the following command:

cd Photos.

If you want to switch to a completely new directory, for example, /home/username/Movies, you have to enter cd followed by the directory's absolute path:

cd /home/username/Movies

Here are some shortcuts to help you navigate:

cd ~[username] goes to another user's home directory.

cd.. moves one directory up.

cd- moves to your previous directory.

cd / moves to root

Post Lab Exercise:

1. Review Shell Command list.
2. Practice today's lab.

SESSION 3: INTRODUCTION TO LINUX TOOLS and DISCUSSION ABOUT COURSE PROJECTS

Intended Learning Outcome:

To study of more UNIX Commands, Discussion about course projects

Expected skills:

a. Ubuntu windows ready for command,

Tools Required:

a. Ubuntu Operating System

Session Detail:

Syn->Syntax
Session Detail:

During this course, you have to do a project. I am proposing a list of possible projects. Following are the rules for the project:

1.Make a Team! You need to make a team consisting on maximum 3 students.

2.Select a Project! Choose a project and discuss.

3.Start the Work! Finalize requirement specification and update on your project site.

4.Be on Time! During the semester, you have to meet several deadlines for the project. Make sure you meet the deadlines.

5.Go Live! You should post deliverables on your project website.

6.Credit! Make sure to give proper credit to the people/resources used during the project

date

–used to check the date and time

Syn:$date

Format	Purpose	Example	Result
+%m	To display only month	$date+%m	06
+%h	To display month name	$date+%h	June
+%d	To display day of month	$date+%d	O1
+%y	To display last two digits of years	$date+%y	09
+%H	To display hours	$date+%H	10
+%M	To display minutes	$date+%M	45
+%S	To display seconds	$date+%S	55

cal

–used to display the calendar
Syn:$cal 2 2009

echo
–used to print the message on the screen.
Syn:$echo "text"

lp
–used to take printouts
Syn:$lp filename
man
–used to provide manual help on every UNIX commands .
Syn:$man unix command $man cat
who & **whoami** –it displays data about all users who have logged into the system currently. The next command displays about current user only. Syn: $ who $whoami
uptime
–tells you how long the computer has been running since its last reboot or power-off. Syn:$uptime
uname
–it displays the system information such as hardware platform, system name and processor, OS type. Syn:$uname–a
hostname
–displays and set system host name
Syn: $ hostname
bc
–stands for „best calculator"

$bc	$ bc		$ bc	$ bc
10/2*3	scale =1		ibase=2	sqrt(196)
15	2.25+1		obase=16	14 quit
	3.35		11010011	
	quit		89275	
			1010	
			Ā	
			Quit	
$bc		$ bc-l		
for(i=1;i<3;i=i+1)I		scale=2		
1		s(3.14)		
2		0		
3 quit				

DISCUSSION ABOUT COURSE PROJECTS

Intended Learning Outcome:

Project Ideas

1. Multi-threaded Web Crawler: Implement a multi-threaded web crawler. The crawler should be able to remember the last URLs and able to resume. Your program should be able to create appropriate number of threads.
2. Process Manager: Identify the system and user processes. For each process provide CPU, memory, and I/O utilization. You should also tag a process as a CPU bound or I/O bound.
3. System Resource Monitor: Identify current available and utilized resources of the system e.g., CPU, memory, I/O, and bandwidth. Your program should be able to log historical system resources and capable to show resource utilization graph.
4. Repository/Directory Synchronizer: Client-server application which is capable to synchronize the local changes to a remote folder. Check drop-box functionality.
5. Task Manager (Android): Check Android Task Manager, you need to implement similar app with addition to allow user to terminate a specific process automatically. The task manager should also identify CPU utilization for each process.
6. Multi-threaded Proxy Server and Client: Develop a multi-threaded proxy server and client. The server accepts any URL from the client, fetches output, and returns to the client. The client should be able to save the output into HTML page with an appropriate name.
7. Power Consumption and Activity Monitor (Android): Develop an application which is able to identify the tasks and activities consuming battery resources. The app should be able to log the historical monitoring data and make it available for end-user to review it.
8. Android Logger: Develop an application to log all user activities in the cell phone and sync the log file to the user's Dropbox account.

Post Lab Exercise:

1. Review Shell Commands.
2. Creating Group for project and selection of project title

Session 4

FILE MANIPULATION COMMANDS
Intended Learning Outcome:

To study of more UNIX Commands, file creation and changing the file permissions

Expected skills:

a. Knowledge about Ubuntu OS and executing basic command in Ubuntu terminal

Tools Required:

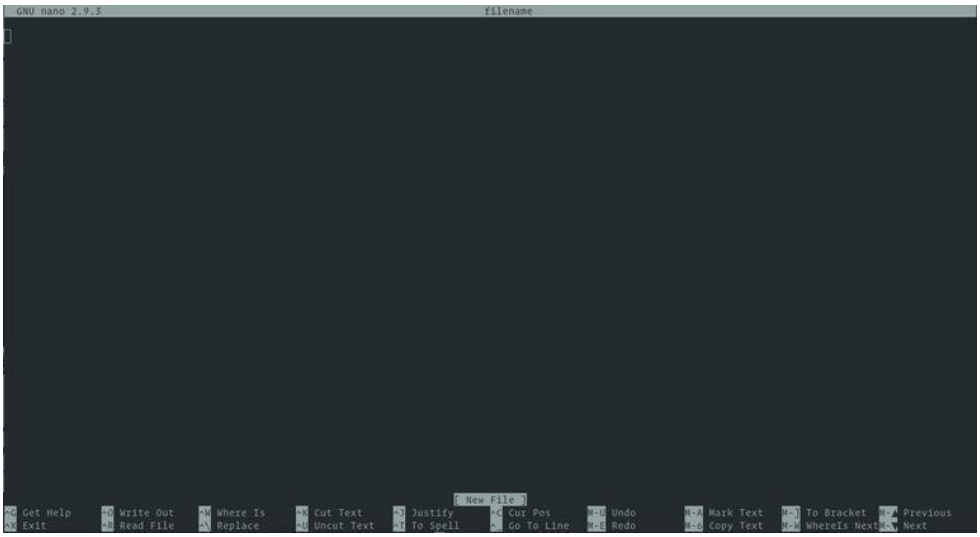
a. Ubuntu Operating System

Session Detail:

File creation using different command and executing some shell command

Cat command –this create, view and concatenate files.
Creation:
Syn: $cat>filename
Viewing:
Syn: $cat filename
Add text to an existing file:
Syn: $cat>>filename
Concatenate:
Syn: $cat file1 file2>file3
$cat file1 file2>>file3 (no over writing of file3)
Nano text editor
This can create and edit files
Nano text editor is pre-installed in most Linux distros.
To check if it is installed on your system type:
nano –version
Install nano into Ubuntu
sudo apt-get install nano

To open an existing file or to create a new file, type nano followed by the file name:
$nano filename



This opens a new editor window, and you can start editing the file. At the bottom of the window, there is a list of the most basic command shortcuts to use with the nano editor.

All commands are prefixed with either ^ or M character. The caret symbol (^) represents the Ctrl key. For example, the ^J commands mean to press the Ctrl and J keys at the same time. The letter M represents the Alt key.

Touch command–used to create a blank file. It also used to create, change and modify timestamps of a file
Syn: $touch file names
Touch command options

- **-a**, change the access time only
- **-c**, if the file does not exist, do not create it
- **-d**, update the access and modification times
- **-m**, change the modification time only
- **-r**, use the access and modification times of the file
- **-t**, creates a file using a specified time

grep–used to search a particular word or pattern related to that word from the file.
$grep search word filename
Eg: $grep anu student
rm–deletes a file from the file system
$rm filename
cp–copies the files or directories
Syn: $cpsource file destination file
Eg:$cp student stud

mv–to rename the file or directory
$mv old file new file

$mv–i student student list(-i prompt when overwrite)

cut–it cuts or pickup a given number of character or fields of the file.

$cut<option><filename>
$cut –c filename
$cut–c1-10emp
$cut–f 3,6emp
$ cut –f 3-6 emp
-c cutting columns
-f cutting fields

head–displays10 lines from the head(top)of a given file

Syn:$head filename
Eg:$head student
To display the top two lines:
……
Syn: $head-2student

tail–displays last 10 lines of the file

$tail filename
$tail student
To display the bottom two lines;
$ tail -2 student

Post Lab Exercise

1. Create files using different command, edit them and practice different Shell Command list.

Session 5: Unix / Linux - File Permission / Access Modes

Intended Learning Outcome:

Checking the file permission and changing the file permissions

Expected skills:

a. Knowledge about Ubuntu OS and executing basic command in Ubuntu terminal

b. Creating file in Ubuntu

Tools Required:

a. Ubuntu Operating System

Session Details:

Unix / Linux - File Permission / Access Modes

File ownership is an important component of Unix that provides a secure method for storing files. Every file in Unix has the following attributes −

- **Owner permissions** − The owner's permissions determine what actions the owner of the file can perform on the file.
- **Group permissions** − The group's permissions determine what actions a user, who is a member of the group that a file belongs to, can perform on the file.
- **Other (world) permissions** − The permissions for others indicate what action all other users can perform on the file.

The permissions are broken into groups of threes, and each position in the group denotes a specific permission, in this order:

read (r), write (w), execute (x)

Checking current permission

$ls –l

it displays various information related to file permission as follows –

```
$ls -l /home/amrood
-rwxr-xr--  1 amrood   users 1024  Nov 2 00:10  myfile
drwxr-xr--- 1 amrood   users 1024  Nov 2 00:10  mydir
```

Here, the first column represents different access modes, i.e., the permission associated with a file or a directory.

- The first three characters (2-4) represent the permissions for the file's owner. For example, **-rwxr-xr--** represents that the owner has read (r), write (w) and execute (x) permission.
- The second group of three characters (5-7) consists of the permissions for the group to which the file belongs. For example, **-rwxr-xr--** represents that the group has read (r) and execute (x) permission, but no write permission.
- The last group of three characters (8-10) represents the permissions for everyone else. For example, **-rwxr-xr--** represents that there is **read (r)** only permission.

Changing Permissions

To change the file or the directory permissions, we can use the **chmod** (change mode) command. There are two ways to use chmod — **the symbolic mode** and **the absolute mode.**

Using chmod in Symbolic Mode

Syn: $chmod category operation permission file

Where,

Category–is the user type

Operation–is used to assign or remove permission

Permission–is the type of permission

File–are used to assign or remove permission all

Examples:

$chmod u-wx student

Removes write and execute permission for users

$ch mod u+rw, g+rw student

Assigns read and write permission for users and groups
$chmod g=rwx student
Assigns absolute permission for groups of all read, write and execute permissions

Category	Operation	Permission
u– users g–group o– others	+assign -remove =assign absolutely	r– read w– write x-execute

Using chmod in absolute Mode

Syn: chmod option for (owner group others) file name

Ex: chmod 764 file1

It means owner has all permission, group has read and write permission and others has read permission

If permission then 1 otherwise 0

	Owner			Group			Others		
	R	W	X	R	W	X	R	W	X
	1	1	1	1	1	0	1	0	0
chmod	7			6			4		

Thus following table shows, and the total of each set of permissions provides a number for that set.

Number	Octal Permission Representation	Ref
0	No permission	---
1	Execute permission	--x
2	Write permission	-w-
3	Execute and write permission: 1 (execute) + 2 (write) = 3	-wx
4	Read permission	r--
5	Read and execute permission: 4 (read) + 1 (execute) = 5	r-x

6	Read and write permission: 4 (read) + 2 (write) = 6	rw-
7	All permissions: 4 (read) + 2 (write) + 1 (execute) = 7	rwx

Post Lab Exercise:

1. Create files, check initial permission and change different permission for different users using both modes.

SESSION 6: INTRODUCTION TO SHELL SCRIPTS

Intended Learning Outcome:

To study of Basic of shell and shell command

Expected skills:

a. Knowledge about Ubuntu OS and executing basic command in Ubuntu terminal

b. Knowledge about file manipulation and changing different permission.

Tools Required:

a. Ubuntu Operating System

Session Detail:

We have seen some basic shell commands, it's time to move on to scripts. There are two ways of writing shell programs. You can type a sequence of commands and allow the shell to execute them interactively. You can store those commands in a file that you can then invoke as a program. This is known as Shell Script. We will use bash shell assuming that the shell has been installed as /bin/sh and that it is the default shell for your login.

Use of Shell Script

1. Shell script can take input from user, file and output them on screen.
2. Useful to create own commands. Save lots of time.
3. To automate some task of day today life.
4. System administration part can be also automated.

How to write and execute?

1. Use any editor to write shell script. The extension is .sh.
2. After writing shell script set execute permission for your script.
   chmod +x script_name
   chmod 764 script_name

3. Execute your scrip. /script_name

Shell Script Format

1. Every script starts with the line
   #!/bin/bash
2. This indicates that the script should be run in the bash shell regardless of which interactive shell the user has chosen.
3. This is very important, since the syntax of different shells can vary greatly.
4. # is used as the comment character.
5. A word beginning with # causes that word and all remaining characters on that line to be ignored.

A Sample Shell Script

```
#!/bin/bash
echo "Hello User"
echo "See the files in current directory
ls
```

Sample Output:
Hello User
See the files in current directory
Folder1, Folder2, File.txt, file1.sh

Post Lab Exercise:

1. Review Shell Script
2. Practice today's lab.
3. Practice some problems based on today's lab.

SESSION 7: SHELL SCRIPTS : Variable, Arithmetic Operation

Intended Learning Outcome:

To study the shell programming

Expected skills:

a. Knowledge about Ubuntu OS and executing basic command in Ubuntu terminal
b. Knowledge about file manipulation and changing different permission.
c. Executing shell program

Tools Required:

a. Ubuntu Operating System

Session Detail:

We have seen some basic shell commands, it's time to move on to scripts. There are two ways of writing shell programs. You can type a sequence of commands and allow the shell to execute them interactively. You can store those commands in a file that you can then invoke as a program. This is known as Shell Script.We will use bash shell assuming that the shell has been installed as /bin/sh and that it is the default shell for your login.

Variables

- In Linux (Shell), there are two types of variable:
  - System variables - created and maintained by Linux itself.
    - **echo $USER**
    - **echo $PATH**
  - User defined variables - created and maintained by user.
- All variables are considered and stored as strings, even when they are assigned numeric values.
- Variables are case sensitive.
- Ex: VAR1, var1 are not same.
- When assigning a value to a variable, just use the name. No spaces on either side of the equals sign.

  var_name=value
- Within the shell we can access the contents of a variable by preceding its name with a ***$***.

myname=A [use quotes if the value contains spaces]
myos=Linux
text = 1+2
echo Your name:$myname
Output: A
echo Your os:$myos
Output: Linux
echo $text
Output: 1+2

- If you enclose a $variable expression in double quotes, it's replaced with its value when the line is executed.

- If you enclose it in single quotes, no substitution takes place. You can also remove the special meaning of the $ symbol by prefacing it with a \.

myvar=”Hello”
echo $myvar [Hello]
echo “$myvar” [Hello]
echo ‘$myvar’ [$myvar]
echo \$myvar [$myvar]

Read

- To read user input from keyboard and store it into a variable use ***read var1,var2,.....varn***

#!/bin/bash
echo -n "Enter your name:”
read name
echo -n "Enter your student no:”
read stdno
echo "Your Name: $name”
echo "Your Age: $stdno”
Sample Output:
Enter your name HH
Enter your student no 1450
Your Name: HH
Your Age: 1450

Shell Arithmetic

Expression Evaluation	Description
`expr1 | expr2`	`expr1` if `expr1` is nonzero, otherwise `expr2`
`expr1 & expr2`	Zero if either expression is zero, otherwise `expr1`
`expr1 = expr2`	Equal
`expr1 > expr2`	Greater than
`expr1 >= expr2`	Greater than or equal to
`expr1 < expr2`	Less than
`expr1 <= expr2`	Less than or equal to
`expr1 != expr2`	Not equal
`expr1 + expr2`	Addition
`expr1 - expr2`	Subtraction
`expr1 * expr2`	Multiplication
`expr1 / expr2`	Integer division
`expr1 % expr2`	Integer modulo

Shell arithmetic example:

```
#!/bin/bash
echo  "Arithmetic operation"
echo " Enter a number "
read n1
echo " Enter another number"
read n2
sum=$(($n1+$n2))
echo " Summation $sum"
sum1=`expr $n1 +$n2`
echo " Summation by another structure $sum1 "
```

Sample Input/Output:

```
Arithmetic operation
Enter a number
10
Enter another number
20
Summation 30
Summation by another structure 30
```

[Note: no space before or after "=" and for 2nd structure use * for multiplication]

Post Lab Exercise:

1. Practice today's lab.
2. Practice different arithmetic operations

SESSION 8: SHELL SCRIPTS- Conditional Statement

If-Else, Case

Intended Learning Outcome:

To study the shell programming

Expected skills:

a. Knowledge about Ubuntu OS and executing basic command in Ubuntu terminal

b. Knowledge about file manipulation and changing different permission.

c. Executing shell program

Tools Required:

a. Ubuntu Operating System

Session Detail:

We have seen some basic shell commands, it's time to move on to scripts. There are two ways of writing shell programs. You can type a sequence of commands and allow the shell to execute them interactively. You can store those commands in a file that you can then invoke as a program. This is known as Shell Script.We will use bash shell assuming that the shell has been installed as /bin/sh and that it is the default shell for your login.

Conditional Statement (If-Else)

if [conditiong1]; then
statement1
elif [condition2]; then
statement2
else statement3
fi

- It is must to put spaces between the [ ]braces and the condition being checked.
- If you prefer putting then on the same line as ***if***, you must add a semicolon to separate the test from the ***then***.
- *It must be end with fi*

String Comparison	Result
`string1 = string2`	True if the strings are equal.
`string1 != string2`	True if the strings are not equal.
`-n string`	True if the string is not null.
`-z string`	True if the string is null (an empty string).

Arithmetic Comparison	Result
`expression1 -eq expression2`	True if the expressions are equal.
`expression1 -ne expression2`	True if the expressions are not equal.
`expression1 -gt expression2`	True if `expression1` is greater than `expression2`.
`expression1 -ge expression2`	True if `expression1` is greater than or equal to `expression2`.
`expression1 -lt expression2`	True if `expression1` is less than `expression2`.
`expression1 -le expression2`	True if `expression1` is less than or equal to `expression2`.
`! expression`	True if the expression is false, and vice versa.

If-Else

```
#!/bin/bash
echo "Enter first number "
read num1
echo "Enter second number"
read num2
if [ $num1 -gt $num2 ] ;  then
echo "$num1 is greater than $num2"
elif [ $num1 -lt $num2 ] ;  then
echo "$num1 is less than $num2"
else
echo "$num1 and $num2 are equal“
fi
```

Sample Input/output:

Enter first number 50
Enter second number 100
100 is greater than 50.

Conditional statement case
Case

case $var in

condition1) statement1 ;;

condition2) statement 2;;

****) statement3;;***

esac

- Notice that each pattern line is terminated with double semicolons ;;.
- You can put multiple statements between each pattern and the next, so a double semicolon is needed to mark where one statement ends and the next pattern begins.
- It must be end with esac

Case

```
#!/bin/sh
   echo “Is it morning? Please answer yes or no” read timeofday
  case “$timeofday” in
    yes) echo “Good Morning”;;
     no ) echo “Good Afternoon”;;
       y ) echo “Good Morning”;;
       n ) echo “Good Afternoon”;;
       * ) echo “Sorry, answer not recognized”;;
   esac
```

Case

```
  #!/bin/sh
  echo “Is it morning? Please answer yes or no” read timeofday
   case “$timeofday” in
   yes | y | Yes | YES ) echo “Good Morning”;;
   n* | N* )      echo “Good Afternoon”;;
  * )     echo “Sorry, answer not recognized”;;
  esac
```

Command Line Arguments

Command line arguments can be passed to the shell scripts. There exists a number of built in variables

$* - command line arguments

$# - number of arguments

$n - nth argument in $*

./script_name arg1 arg2 .... argn

Post Lab Exercise:

1. Practice today’s lab.
2. Practice some problems based on conditional statements(using if else, case)

SESSION 9: SHELL SCRIPTS- LOOP, FOR, WHILE, UNTIL, FUNCTION

Intended Learning Outcome:

To study the shell programming

Expected skills:

a. Knowledge about Ubuntu OS and executing basic command in Ubuntu terminal

b. Knowledge about file manipulation and changing different permission.

c. Executing shell program

Tools Required:

a. Ubuntu Operating System

Session Detail:

We have seen some basic shell commands, it's time to move on to scripts. There are two ways of writing shell programs. You can type a sequence of commands and allow the shell to execute them interactively. You can store those commands in a file that you can then invoke as a program. This is known as Shell Script.We will use bash shell assuming that the shell has been installed as /bin/sh and that it is the default shell for your login.

FOR LOOP

```
for variable in list
  do
     statement
  done

  for (( expr1; expr2; expr3 ))
    do
     statement
    done
[ Need permission before executing script ]
```

While

Structure

```
While condition do
     statements
```

```
#!/bin/bash
password="abc"
echo "Enter password"
read pass
while [ $pass != $password ]
do
echo "Wrong Password, Try again"
read  pass
done
echo "Write Password"
```

Done
Example:

```
#!/bin/bash
i=1
while [ $i –le 10 ]
do
echo “ $i “
done
```

Until

```
Until condition
 do
     statements
done
#!/bin/bash
i=1
until [ $i –gt 10 ]
do
echo “ $i “
done
```

```
#!/bin/bash
password="abc"
echo "Enter password"
read pass
until [ $pass != $password ]
do
   echo "Wrong Password, Try   again"
    read  pass
done
echo "Write Password"
```

Function

1. Functions can be defined in the shell and it is very useful to structure the code.
2. To define a shell function simply write its name followed by empty parentheses and enclose the statements in braces.
   function_name ()
   {statements
   }
3. Function must be defined before one can invoke it.

```
#!/bin/sh
 foo() {
   echo “Function foo is executing”
}
```

echo "script starting"
foo
echo "script ending"

output
script starting
Function foo is executing
script ending

- Be careful :
- Function calling can be recursive.

```
f0 {
  statements f
  }
f
```

- The parameter must be passed every time a function is invoked either from main or from any other functions.

Post Lab Exercise:

1. Practice today's lab.
2. Practice some problems based on loop.
3. Practice some problems using function.

Session 10 and 11: Implementing CPU scheduling algorithms
FCFS and SJF Algorithm

Intended Learning Outcome: To write a C program for implementation of FCFS and SJF scheduling algorithms.

Expected skills:

a. Knowledge about programming language.
b. Executing C or any other language in Ubuntu OS
c. Knowledge about CPU scheduling algorithm

Tools Required:

a. Ubuntu Operating System, C Compiler

ALGORITHM:

Step 1: Inside the structure declare the variables.
Step 2: Declare the variable i,j as integer,totwtime and totttime is equal to zero.
Step 3: Get the value of „n" assign pid as I and get the value of p[i].btime.
Step 4: Assign p[0] wtime as zero and tot time as btime and inside the loop calculate wait time and turnaround time.
Step 5: Calculate total wait time and total turnaround time by dividing by total number of process.
Step 6: Print total wait time and total turnaround time.
Step 7: Stop the program.

PROGRAM:

```
 #include<stdio.h>
#include<stdlib.h>
struct fcfs
{
int pid;
int btime;
int wtime;
int ttime;
}
p[10];
int main()
{ int i,n;
int towtwtime=0,totttime=0;
printf("\n fcfs scheduling...\n");
printf("enter the no of process");
scanf("%d",&n);
for(i=0;i<n;i++)
{
p[i].pid=1;
printf("\n burst time of the process");
scanf("%d",&p[i].btime);
}
p[0].wtime=0;
p[0].ttime=p[0].btime;
totttime+=p[i].ttime;
for(i=0;i<n;i++)
{
p[i].wtime=p[i-1].wtime+p[i-1].btim p[i].ttime=p[i].wtime+p[i].btime;
totttime+=p[i].ttime;
towtwtime+=p[i].wtime;
```

```
}
for(i=0;i<n;i++)
 {{ printf("\n waiting time for process");
 printf("\n turn around time for process");
printf("\n");
 }}
printf("\n total waiting time :%d", totwtime );
printf("\n average waiting time :%f",(float)totwtime/n); printf("\n total turn around time :%d",totttime);
printf("\n average turn around time: :%f",(float)totttime/n);
}
```

Implementing SJF Algorithm

AIM: To write a C program for implementation of SJF scheduling algorithms.

ALGORITHM:

Step 1: Inside the structure declare the variables.

Step 2: Declare the variable i,j as integer,totwtime and totttime is equal to zero.

Step 3: Get the value of „n‟ assign pid as I and get the value of p[i].btime.

Step 4: Assign p[0] wtime as zero and tot time as btime and inside the loop calculate wait time and turnaround time.

Step 5: Calculate total wait time and total turnaround time by dividing by total number of process.

Step 6: Print total wait time and total turnaround time. Step

7: Stop the program.

Implementing Priority Scheduling Algorithm

AIM: To write a C program for implementation of Priority scheduling algorithms.

ALGORITHM:

Step 1: Inside the structure declare the variables.

Step 2: Declare the variable i,j as integer, totwtime and totttime is equal to zero.

Step 3: Get the value of „n‟ assign p and allocate the memory.

Step 4: Inside the for loop get the value of burst time and priority.

Step 5: Assign wtime as zero .

Step 6: Check p[i].pri is greater than p[j].pri .

Step 7: Calculate the total of burst time and waiting time and assign as turnaround time.

Step 8: Stop the program.

Implementing Round Robin Scheduling Algorithm

AIM: To write a C program for implementation of Round Robin scheduling algorithms.
ALGORITHM:

Step 1: Inside the structure declare the variables.
Step 2: Declare the variable i,j as integer, totwtime and totttime is equal to zero.
Step 3: Get the value of „n‟ assign p and allocate the memory.
Step 4: Inside the for loop get the value of burst time and priority and read the time quantum.
Step 5: Assign wtime as zero.
Step 6: Check p[i].pri is greater than p[j].pri .
Step 7: Calculate the total of burst time and waiting time and assign as turnaround time.
Step 8: Stop the program.

Post Lab Experiments:

1. **Practice FCFS algorithm using C or any other language and check the result for different values**
2. **Practice SJF algorithm using C or any other language and check the result for different values**
3. **Practice Priority scheduling algorithm using C or any other language and check the result for different values**
4. **Practice Round Robin algorithm using C or any other language and check the result for different values**