Basic Linux Commands:

PWD: Whenever you feel lost in the filesystem, call the PWD command to know where you are:

Syntax: PWD

It will print the current folder path.

ls: This command is used to show the list of existing directories.

Syntax: ls

Clear: This command is used to clear the screen on the terminal.

Syntax: Clear

History: This command is used to show the list of all activities that is performed on the terminal.

Syntax: History

Date: This command is used to show the date and time of the system.

Syntax: date

Cd: This command is used for two different purpose

(i) use 1: The command is used to change the directory.

Syntax: cd <directory-name>

Example: cd Desktop

(ii) use 2: This command is also used to go to the present directory.

Syntax: cd..

mkdir: This command is used created a directory.

Syntax: mkdir <directory-name>

Example: mkdir ankita

rmdir: Just as you can create a folder using mkdir you can remove a folder using rmdir:

Syntax: rmdir <existing-directory-name>

Eg: rmdir ankita

rm: This command is used to delete an existing file in the directory.

Syntax: rm <existing-file-name>

example: rm ank.txt

cp: This command is used to copy the contents of a file to another file2.

Syntax: cp file1 file2

NOTE: The above syntax means the contents of file1 is copied to file2.

Eg: cp ank.text ankita.txt

MV: mv stands for move. MV is used to move one or more files or directories from one place to another in a file system.

Syntax: mv [options] source destination

open: The open commands let you open a file using this

Syntax: open <filename>

The same command can also be used to open an application.

Syntax: open <application name>

touch: This command is used to create an emptyfile in the directory.

Syntax: touch <new-file-name>

ls-l: It is used to summarize all most important information about the file on one line.

Syntax: ls-l

Cat: This command is used to create an empty file & to show the contents of the file.

(i) TO create an empty file:

Syntax: cat > (filename)

Eg: cat > ank.txt

NOTE: After entering the contents, "Ctrl+D" is pressed to save and exit.

(ii) TO show the contents of a file

Syntax: cat <existing-file-name>

Eg: cat ank.txt

Chmod: The command is used to change the permission of a file.

Eg: chmod 777 ankita.txt.

NOTE: Here first digit represents the degree of permission for owner and second and third represents the permission for group and other users. Each degree is the sum of r(read), write(w) & execute(x), each having value, r=4, w=2, x=1

ln: The ln command is part of the linux file system commands.

It is used to create links.

What is a link?

→ A link is a connection between a file name and the actual data on the disk.

→ There are two main types of links that can be created. i) Hard links & soft links

Hard links:

A hard link is created using

ln <original> <link>

Soft links:
A soft links using the [-s] option of [ln]:
ln-s <original> <links>

whoami: This command is used to show the name of the user
Syntax: whoami

Uname: This command is used to show the name of the operating system.
Syntax: Uname

PS-ef: This command is used to show the list of processes.
Syntax: PS-ef

nano: Nano is a beginner friendly editor
Run it using nano <file name>

who: The who command display the users logged into the system.

Sudo: Sudo is commonly used to run a command as root.

Passwd: You can change the password using the [Passwd] command.
Syntax: passwd <username> <new password>

head: The command is used to print line from first.
Eg: head -2 mane
first two lines are printed.

Tail: This command is used to print lines from last.
Eg: tail -2 mane
last two lines are printed.

Experiment: Shell script to add two numbers

Program:

```
echo "Enter 2 numbers"
read a b
echo $a
echo $b
Sum=$((a+b))
echo $ sum
```

output:

```
Enter 2 numbers
10 210
10
210
220
```

Experiment: Shell script to substract two numbers.

program:

```
echo "enter 2 numbers"
read a b
sub = ' expr $a - $b'
echo $ sub
```

output:

```
Enter 2 numbers
12 5
7
```

FCFS CPU SCHEUDLING:

experiment : program to implement fcfs.

program:

```
#include<stdio.h>
int main()
{
 int TO[10], BT[10], T1[10], TAT[10], i, n, sum;
 float average;
 printf("Enter the number of processes:");
 scanf("%d", &n);
 printf("Enter the arrival time in ascending manner");
 for(i=0; i<n; i++)
  {
   printf("Enter the arrival time of P[%d]=", i);
   scanf("%d", &TO[i]);
  }
 for(i=0; i<n; i++)
  {
   printf("Enter the burst time of P[%d]=", i);
   scanf("%d", &BT[i]);
  }
 sum = TO[0];
 for(i=0; i<n; i++)
  {
   sum = sum + BT[i];
   T1[i] = sum;
   printf("The completion time of P[%d] = %d", i, T1[i]);
  }
for(i=0; i<n; i++)
  {
   TAT[i] = T1[i] - TO[i];
   printf("\n the turn arround time of P[%d] = %d", i, TAT[i]);
  }
 sum = 0;
 for(i=0; i<n; i++)
  {
   sum = sum + TAT[i];
  }
 average = (float) sum/n;
 printf("\n the average turn around time is = %2f", average);
 return 0;
}
```

Output:-

```
Enter the number of process : 3
Enter the arrival time in ascending manner.
Enter the arrival time of P[0] = 0
Enter the arrival time of P[1] = 1
Enter the arrival time of P[2] = 2
Enter the burst time of P[0] = 5
Enter the burst time of P[1] = 4
Enter the burst time of P[2] = 3
The completion time of P[0] = 5
The completion time of P[1] = 9
The completion time of P[2] = 12
The turn arround time of P[0] = 5
The turn arround time of P[1] = 8
The turn arround time of P[2] = 10
The average turn arround time is = 7.67
```

SJF scheduling

experiment: Program to implement SJF for processes having equal arrival time.

program:

```
#include <stdio.h>
void sorting(int[], int);
{
int to=0, bt[10], t1[10], tat[10], c[10], i,n,j,sum;
float average;
printf("enter the number of processes");
scanf("%d", &n);
for(i=0; i<n; i++)
{
printf("enter the burst time of P[%d] = "; i);
scanf("%d", &bt[i]);
}
for(i=0; i<n; i++)
{
c[i] = bt[i];
}
sorting(c,n);
sum = to;
for(i=0; i<n; i++)
{
for(j=0; j<n; j++)
{
if(c[i] == bt[j])
{
sum = sum + bt[j];
t1[j] = sum;
}
}
}
}
```

```
for(i=0; i<n; i++)
{
printf("\n the final time of P[%d] = %d", i, t1[i]);
}
for(i=0; i<n; i++)
{
tat[i] = t1[i] - to;
printf("\n the tat of P[%d] = %d", i, tat[i]);
}
sum = 0;
for(i=0; i<n; i++)
{
sum = sum + tat[i];
}
average = (float) sum/n;
printf("\n The average turn around time is %
%.2f", average);
return 0;
}
void sorting(int P[], int q)
{
int t, r, s;
for(r=0; r<q; r++)
{
for(s=r+1; s<q; s++)
{
if(P[r] > P[s])
{
t = P[r];
P[r] = P[s];
P[s] = t;
}
}
}
}
```

Output:-

```
Enter number of processes = 4
Enter the burest time of P[0] = 6
Enter the burest time of P[1] = 8
Enter the burest time of P[2] = 7
Enter the burest time of P[3] = 3

The final time of P[0] = 9
The final time of P[1] = 24
The final time of P[2] = 16
The final time of P[3] = 3

The tat of P[0] = 9
The tat of P[1] = 24
The tat of P[2] = 16
The tat of P[3] = 3
The average teern around time is 13.00
```

Experiment: program to implement round robin algorithm for processes having equal arrival time

program:

```
#include<stdio.h>
int i,j;
int greaterc(int[], int);
int main()
{
int at=0, bt[20], t1[20], tat[20], t, sum=0, n, k;
float avg;
printf("enterc the number of process:");
scanf("%d", &n);
printf("enterc the value of the time quantum:");
scanf("%d", &t);
printf("enterc the burst time of processes:\n");
for(i=0;i<n;i++)
{
  printf("bt[%d] =", i);
  scanf("%d", &bt[i]);
}
k= greaterc(bt,n);
printf("The index of process having greaterc burst
        time is %d\n", k);
printf("\n The ready q is given below:\n");
while(bt[k]!=0)
{
 for(i=0,i<n;i++)
  {
    if(i=0;i<n;i++)
     {
       if(bt[i]>t)
       {
        bt[i]= bt[i]-t;
        sum= sum+t;
```

```
            t1[i] = sum;
            printf("\np[%d]=%d\t", i, t1[i]);
        }
    }
}
else if (bt[i]>0 && bt[i]<=t)
{
    sum = sum + bt[i];
    t1[i] = sum;
    bt[i] = 0;
    printf("\np[%d]=%d\t", i, t1[i]);
}
printf("\n\n the final time of processes are:\n");
for (i=0; i<n; i++)
{
    printf("\n t1[%d] = %d\n", i, t1[i]);
}
sum = 0;
printf("\n the turen around time of processes are:\n");
for (i=0; i<n; i++)
{
    tat[i] = t1[i] - at;
    printf("\n tat[%d] = %d\n", i, tat[i]);
    sum = sum + tat[i];
}
avg = (float) sum/n;
printf("\n The avereage turen around time is
        %2f\n", avg);
return 0;
}
```

```
int greater(int p[], int n)
{
    int big = p[0], end = 0;
    for (i=1; i<n; i++)
    {
        if (big < p[i])
        {
            big = p[i];
            end = i;
        }
    }
    return end;
}
```

output:

Enter the number of process : 3
Enter the value of time quantum : 4
Enter the burst time of processes:
bt[0] = 5
bt[1] = 7
bt[2] = 8

the index of processing having greater burst time is 2.

the ready q is given below:

p[0] = 4 p[1] = 8 p[2] = 12 p[0] = 13 p[1] = 16 p[2] = 20

The final time of process are:

t1[0] = 13
t1[1] = 16
t1[2] = 20

The turen around time of processes are:

tat[0] = 13
tat[1] = 16
tat[2] = 20

The average turen around time is 16.33

Experiment: Program to implement FIFO algorithm

Program:

```
#include<stdio.h>
int grt(int [],int);
int main()
{
int i,j,n,pf=0,p[20],ms[20],rest[20],refb[20],m,
age[20],k=0,t,flag,b,h,c;
printf("Enter the number of pages:");
scanf("%d",&n);
for(i=0;i<n;i++)
{
p[i]=[i];
age[i]=-1;
rest[i]=0;
}
printf("Enter the size of memory space:");
scanf("%d",&m);
for(i=0;i<m;i++)
{
ms[i]=-1;
}
printf("Enter the size of reference set:");
scanf("%d",&rc);
printf("\n Enter the references:\n");
for(i=0;i<rc;i++)
{
printf("reference no(%d)=",i);
scanf("%d",&t);
if(t>=0&&t<n)
refb[i]=t;
else
{
printf("please don't exceed the page limit----\n");
i--;
}
}
printf("\n\nwere references are:\n");
for(i=0;i<rc;i++)
printf("%d",refb[i]);
for(i=0;i<rc;i++)
{
flag=0;
for(j=0;j<m;j++)
{
if(refb[i]==ms[j])
flag=1;
}
if(flag==0)
{
pf++;
if(k<m)
{
ms[k++]=refb[i];
}
else
{
h=grt(age,n);
for(b=0;b<m;b++)
{
if(ms[b]==h)
{
ms[b]=refb[i];
age[h]=-1;
printf("\n process %d replaced process
%d in the iteration %d\n",refb[i],
h,i+1);
}}}
```

```
    for (c=0; c<m; c++)
      {
        if (ms[c] != -1)
        age[ms[c]]++;
      }
  }
  printf("\n Total number of page fault (S) = %d", PF);
  return 0;
}
int get (int P[], int q)
{
  int a, big = P[0], ind = 0;
  for (a=0; a<q; a++)
  {
    if (big < P[a])
    {
      big = P[a];
      ind = a;
    }
  }
  return ind;
}
```

output:

1) Enter the number of pages = 4
enter the size of memory space = 3
Enter the size of reference bit: 5
enter the references:
reference no.(0) = 0
reference no.(1) = 0
reference no.(2) = 1
reference no.(3) = 2
reference no.(4) = 3
process 3 replaced process 0 in the iteration 5
Total number of page fault(S) = 4

2) Enter the number of pages : 4
Enter the size of memory space : 3
Enter the size of reference bit: 5
Enter the references:
reference no.(0) = 1
reference no.(1) = 0
reference no.(2) = 2
reference no.(3) = 0
reference no.(4) = 1
Your references are:
1 0 2 0 1
Total number of page fault(S) = 3

3) Enter the number of pages = 3
Enter the size of memory space : 3
Enter the size of reference bit : 9
enter the references
references no.(0) = 0
references no.(1) = 3
Please don't exceed the page limit....
reference no.(1) = 1
reference no.(2) = 2
reference no.(3) = 0
reference no.(4) = 1
reference no.(5) = 0
reference no.(6) = 1
reference no.(7) = 0
reference no.(8) = 2
Your references are:
0 1 2 0 1 0 1 0 2
Total number of page fault(S) = 3