Lab 1: To get acquainted with LINUX commands.

AIM: To study and execute the commands in unix.

COMMANDS:
1.Date Command:
This command is used to display the current data and time.
Syntax:
$date
$date +%ch
Options:
D = Day in „mm/dd/yy" format
P = Display AM or PM
Y = Display the full year.
Z = Time zone,
To change the format:
Syntax: $date '+%H-%M-%S'

2.Calendar Command:
This command is used to display the calendar of the year or the particular month of calendar year.
Syntax:
a.$cal <year>
b.$cal <month> <year>
Here the first syntax gives the entire calendar for given year & the second Syntax gives the calendar of reserved month of that year.

3.Echo Command:
This command is used to print the arguments on the screen .
Syntax: $echo <text>
Multi line echo command: To have the output in different line, the following command can be used.
Syntax:
$echo "text
>line2
>line3"

4.Banner Command:
It is used to display the arguments in '#' symbol .
Syntax: $banner <arguments>

5.'who' Command:
It is used to display who are the users connected to our computer currently.
Syntax: $who – options
Options: -
H–Display the output with headers.
b–Display the last booting date or time or when the system was lastely rebooted.

6.'who am i' Command:
Display the details of the current working directory.
Syntax: $whoami

7.'tty' Command:
It will display the terminal name.
Syntax: $tty

8.'Binary' Calculator Command:
It will change the '$' mode and in the new mode, arithematic operations such as +,- ,*,/, %,sqrt(),length(),=, etc can be performed. This command is used to go to the binary calculus mode.
Syntax:
$bc operations

9.'CLEAR' Command:
It is used to clear the screen.
Syntax: $clear

10.'MAN' Command:
It help us to know about the particular command and its options & working. It is like 'help' command in windows.
Syntax: $man <command name>

11.LIST Command :
It is used to list all the contents in the current working directory.
Syntax: $ ls – options <arguments>
If the command does not contain any argument means it is working in the Current directory.
Options:
a– used to list all the files including the hidden files.
c– list all the files column wise.
m- list the files separated by commas.
p- list files include „/" to all the directories.
r- list the files in reverse alphabetical order.
flist the files based on the list modification date.
x-list in column wise sorted order.

DIRECTORY RELATED COMMANDS:
1.Present Working Directory Command:
To print the complete path of the current working directory.
Syntax: $pwd

2.MKDIR Command :
To create or make a new directory in a current directory .
Syntax: $mkdir <directory name>

3.CD Command :
To change or move the directory to the mentioned directory .
Syntax: $cd <directory name.

4.RMDIR Command :
To remove a directory in the current directory & not the current directory itself.
Syntax: $rmdir <directory name>

FILE RELATED COMMANDS
1.CREATE A FILE:
To create a new file in the current directory we use CAT command.
Syntax:
$cat > <filename>.
The > symbol is redirectory we use cat command.

2.DISPLAY A FILE:
To display the content of file mentioned we use CAT command without '>' operator.
Syntax: $cat <filename>.
Options –s = to neglect the warning /error message.

3.COPYING CONTENTS:
To copy the content of one file with another. If file doesnot exist, a new file is created and if the file exists with some data then it is overwritten.
Syntax:
$ cat <filename source> >> <destination filename>

4.SORTING A FILE:
To sort the contents in alphabetical order in reverse order.
Syntax: $sort <filename >
Option: $ sort –r <filename>

5.COPYING CONTENTS FROM ONE FILE TO ANOTHER :
To copy the contents from source to destination file . so that both contents are same.
Syntax:
$cp <source filename> <destination filename>
$cp <source filename path > <destination filename path>

6.MOVE Command:
To completely move the contents from source file to destination file and to remove the source file.
Syntax: $ mv <source filename> <destination filename>

7.REMOVE Command:
To permanently remove the file we use this command.
Syntax: $rm <filename>

8.WORD Command:
To list the content count of no of lines, words, characters.
Syntax:
$wc<filename>
Options:
-c – to display no of characters.
-l – to display only the lines.-w – to display the no of words.

Lab 2: SYSTEM CALLS

Task 1: PROCESS CREATION

ALGORITHM:

STEP 1: Start the program.
STEP 2: Declare pid as integer.
STEP 3: Create the process using fork( ) function call.
STEP 4: Check pid is less than 0 then print error
else if pid is equal to 0 then execute command
else parent process wait for child process.
STEP 5: Stop the program.

PROGRAM:

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
int main()
{
        int pid;
        pid = fork();
        if(pid == 0)
        {
                printf("Child\n");
                printf("Parent Process PID is %d \n",getppid());
                printf("Child Process PID is %d \n",getpid());
        }
        else
        {
                printf("Parent\n");
                printf("Parent Process PID is %d \n",getpid());
                printf("Parent's Parent Process PID is %d \n",getppid());
        }
        return 0;
}
```

Task 2: SLEEP COMMAND
ALGORITHM:
STEP 1: Start the program.
STEP 2: Create process using fork and assign into a variable.
STEP 3: If the value of variable is < 0 print not created and > 0 process created and else print child created.
STEP 4: Create child with sleep of 2.
STEP 5: Stop the program.

PROGRAM:

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
int main()
{
      int id;
      id = fork();
      if(id < 0)
      {
            printf("Cannot create the process\n");
            exit(1);
      }
      else if(id == 0)
      {
            sleep(2);
            printf("Hi, I am from Child\n");
      }
      else
      {
            printf("Hi, I am from Parent\n");
            exit(1);
      }
       printf("\n");
       return 0;
}
```

Lab 3: I/O System Calls

Task 1: READING FROM A FILE

ALGORITHM:

1.Get the data from the user.
2.Open a file.
3.Read from the file.
4.Close the file.

PROGRAM:

```
#include<stdio.h>
int main()
{
        char str[100];
        FILE *fp;
        fp=fopen("file1.dat","r");
        while(!feof(fp))
        {
                fscanf(fp,"%s",str);
                printf(" %s ",str);
        }
        fclose(fp);
}
```

OUTPUT:

Task 2: WRITING INTO A FILE
ALGORITHM:
Step1.Get the data from the user.
Step2.Open a file.
Step3.Write the data from the file.
Step4.Get the data and update the file.

PROGRAM:

```
#include<stdio.h>
int main()
{
        char str[100];
        FILE *fp;
        printf("Enter the string");
        gets(str);
        fp=fopen("file1.dat","w+");
        while(!feof(fp))
        {
                fscanf(fp,"%s",str);
        }
        fprintf(fp,"%s",str);
}
```

Lab 4: FIRST COME FIRST SERVE CPU SCHEDULING

PROBLEM DESCRIPTION:

CPU scheduler will decide which process should be given the CPU for its execution. For this it use different algorithm to choose among the processes. One among those algorithm is FCFS algorithm. In this algorithm the process which arrive first is given the CPU after finishing its request only it will allow CPU to execute other process.

ALGORITHM:

Step1: Create the number of process.
Step2: Get the ID and Service time for each process.
Step3: Initially, Waiting time of first process is zero and Total time for the first process is the starting time of that process.
Step4: Calculate the Total time and Processing time for the remaining processes.
Step5: Waiting time of one process is the Total time of the previous process.
Step6: Total time of process is calculated by adding Waiting time and Service time.
Step7: Total waiting time is calculated by adding the waiting time for lack process.
Step8: Total turn around time is calculated by adding all total time of each process.
Step9: Calculate Average waiting time by dividing the total waiting time by total number of process.
Step10: Calculate Average turn around time by dividing the total time by the number of process.
Step11: Display the result.

PROGRAM

```
#include<stdio.h>
struct process
{
	int id,wait,ser,tottime;
}p[20];
int main()
{
	int i,n,j,totalwait=0,totalser=0,avturn,avwait;
	printf("enter number of process");
	scanf("%d",&n);
	for(i=1;i<=n;i++)
	{
		printf("enter process_id");
		scanf("%d",&p[i].id);
		printf("enter process service time");
```

```
        scanf("%d",&p[i].ser);
    }
    p[1].wait=0;
    p[1].tottime=p[1].ser;
    for(i=2;i<=n;i++)
    {
        for(j=1;j<i;j++)
        {
            p[i].wait=p[i].wait+p[j].ser;
        }
        totalwait=totalwait+p[i].wait;
        p[i].tottime=p[i].wait+p[i].ser;
        totalser=totalser+p[i].tottime;
    }
    avturn=totalser/n;
    avwait=totalwait/n;
    printf("Id\tservice\twait\ttotal");
    for(i=1;i<=n;i++)
    {
        printf("\n%d\t%d\t%d\t%d\n",p[i].id,p[i].ser,p[i].wait,p[i].tottime);
    }
    printf("average waiting time %d\n",avwait);
    printf("average turnaround time %d\n",avturn);\
    return 0;
}
```

OUTPUT

```
enter number of process 4
enter process_id 901
enter process service time 4
enter process_id 902
enter process service time 3
enter process_id 903
enter process service time 5
enter process_id 904
enter process service time 2
Id      service wait    total
901     4       0       4

902     3       4       7

903     5       7       12

904     2       12      14
average waiting time 5
average turnaround time 8
```

Lab 5: SHORTEST JOB FIRST CPU SCHEDULING

PROBLEM DESCRIPTION:

CPU scheduler will decide which process should be given the CPU for its execution. For this it uses different algorithm to choose among the processes. One among those algorithm is SJF algorithm. In this algorithm the process which has least service time is given the CPU and after finishing its request only it will allow CPU to execute other processes.

ALGORITHM:

Step1:Get the number of process.
Step2:Get the id and service time for each process.
Step3:Initially the waiting time of first short process as 0 and total time of first short is process the service time of that process.
Step4:Calculate the total time and waiting time of remaining process.
Step5:Waiting time of one process is the total time of the previous process.
Step6:Total time of process is calculated by adding the waiting time and service time of each process.
Step7:Total waiting time calculated by adding the waiting time of each process.
Step8:Total turn around time calculated by adding all total time of each process.
Step9:calculate average waiting time by dividing the total waiting time by total number of process.
Step10:Calculate average turn around time by dividing the total waiting time by total number of process.
Step11:Display the result.

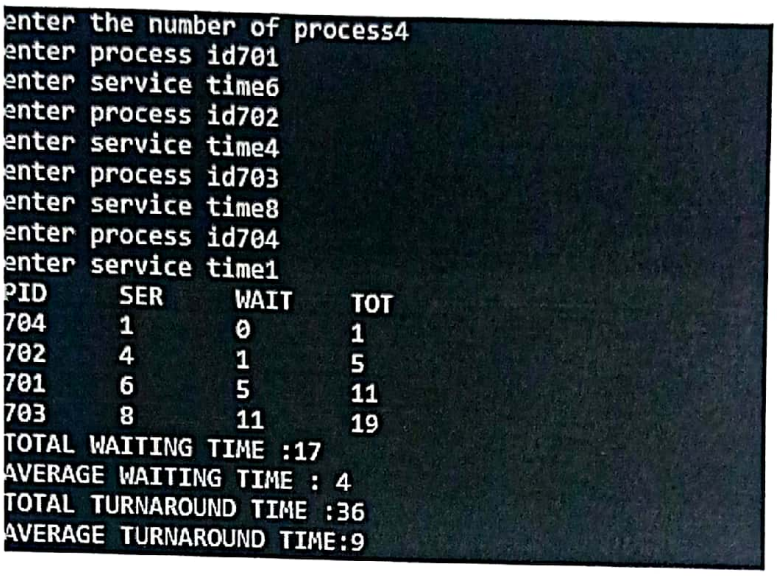
PROGRAM:

```
#include<stdio.h>
struct ff
{
        int pid,ser,wait;
}p[20];
struct ff tmp;
int main()
{
        int i,n,j,tot=0,avwait,totwait=0,tturn=0,aturn;
        printf("enter the number of process");
        scanf("%d",&n);
```

```
        for(i=0;i<n;i++)
        {
                printf("enter process id");
                scanf("%d",&p[i]);
                printf("enter service time");
                scanf("%d",&p[i].ser);
                p[i].wait=0;
        }
        for(i=0;i<n-1;i++)
        {
                for(j=i+1;j<n;j++)
                {
                        if(p[i].ser>p[j].ser)
                        {
                                tmp=p[i];
                                p[i]=p[j];
                                p[j]=tmp;
                        }
                }
        }
        printf("PID\tSER\tWAIT\tTOT\n");
        for(i=0;i<n;i++)
        {
                tot=tot+p[i].ser;
                tturn=tturn+tot;
                p[i+1].wait=tot;
                totwait=totwait+p[i].wait;
                printf("%d\t%d\t%d\t%d\n",p[i].pid,p[i].ser,p[i].wait,tot);
        }
        avwait=totwait/n;
        aturn=tturn/n;
        printf("TOTAL WAITING TIME :%d\n",totwait);
        printf("AVERAGE WAITING TIME : %d\n",avwait);
        printf("TOTAL TURNAROUND TIME :%d\n",tturn);
        printf("AVERAGE TURNAROUND TIME:%d\n",aturn);
}
```

Operating Systems

OUTPUT:



Lab 6: Disk Scheduling using FCFS

DESCRIPTION: One of the responsibilities of the operating system is to use the hardware efficiently. For the disk drives, meeting this responsibility entails having fast access time and large disk bandwidth. Both the access time and the bandwidth can be improved by managing the order in which disk I/O requests are serviced which is called as disk scheduling. The simplest form of disk scheduling is, of course, the first-come, first-served (FCFS) algorithm. This algorithm is intrinsically fair, but it generally does not provide the fastest service.

In the SCAN algorithm, the disk arm starts at one end, and moves towards the other end, servicing requests as it reaches each cylinder, until it gets to the other end of the disk. At the other end, the direction of head movement is reversed, and servicing continues. The head continuously scans back and forth across the disk.

C-SCAN is a variant of SCAN designed to provide a more uniform wait time. Like SCAN, C-SCAN moves the head from one end of the disk to the other, servicing requests along the way. When the head reaches the other end, however, it immediately returns to the beginning of the disk without servicing any requests on the return trip.

PROGRAM
FCFS DISK SCHEDULING ALGORITHM

```
#include<stdio.h>
int main()
{
        int t[20], n, i, j, tohm[20], tot=0;
        float avhm;
        printf("enter the no.of tracks");
        scanf("%d",&n);
        printf("enter the tracks to be traversed\n");
        for(i=2;i<n+2;i++)
                scanf("%d",&t[i]);
        for(i=1;i<n+1;i++)
        {
                tohm[i]=t[i+1]-t[i];
                if(tohm[i]<0)
                        tohm[i]=tohm[i]*(-1);
        }
        for(i=1;i<n+1;i++)
```