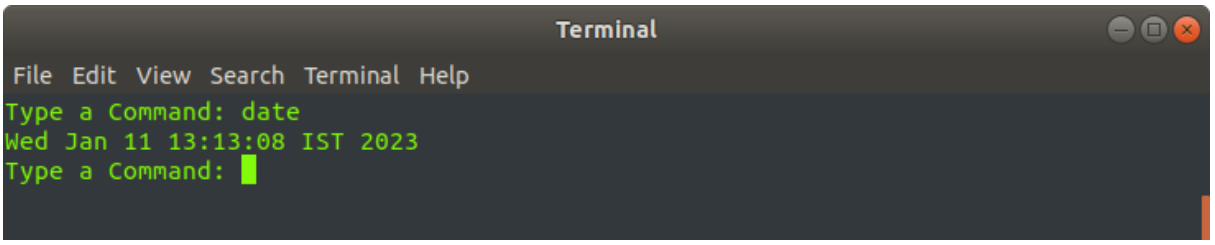
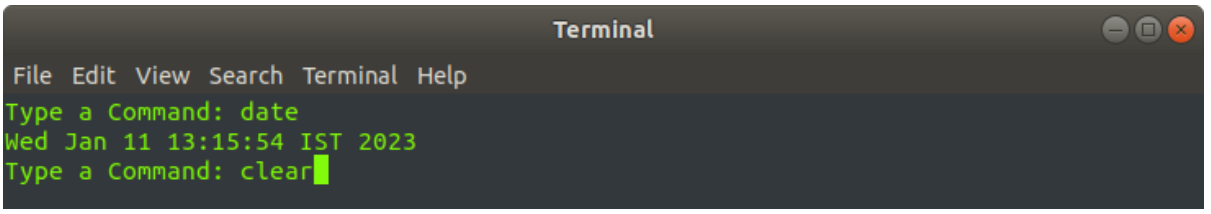
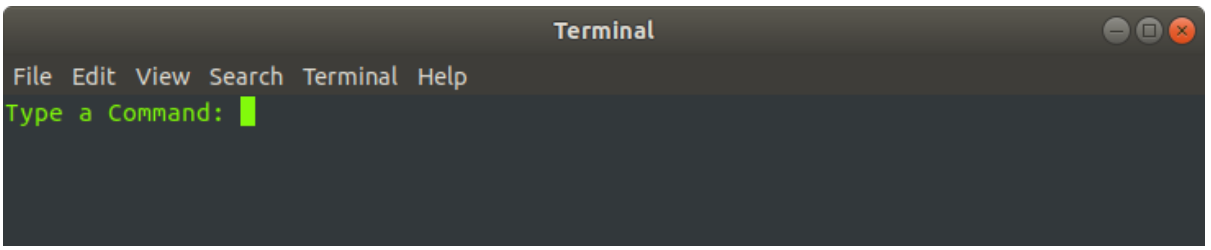
Aim: To study basic linux commands.

1. date - This command prints date and time.

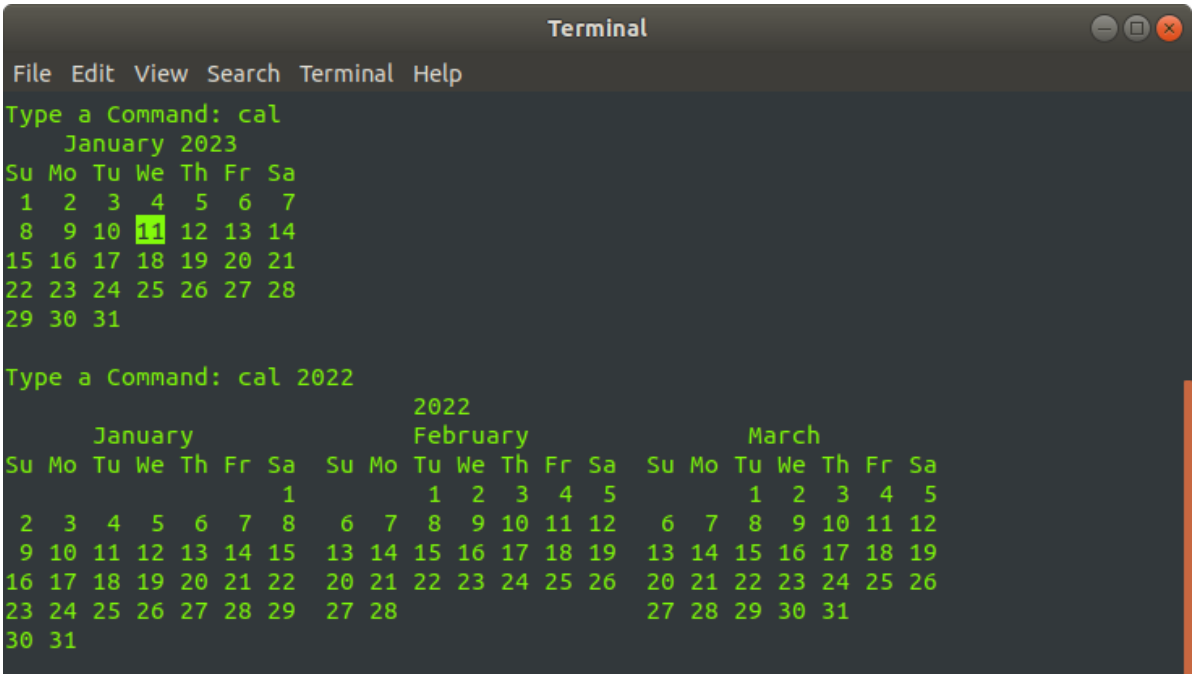


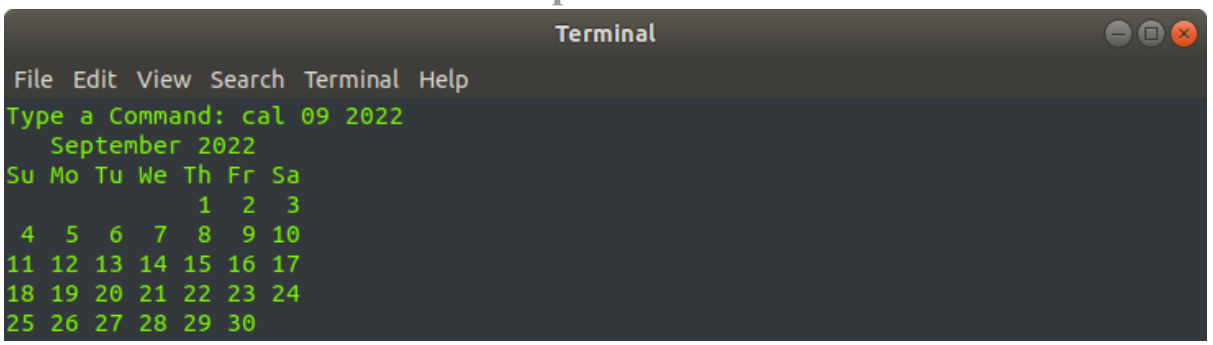
2. clear - This command clears your terminal screen.



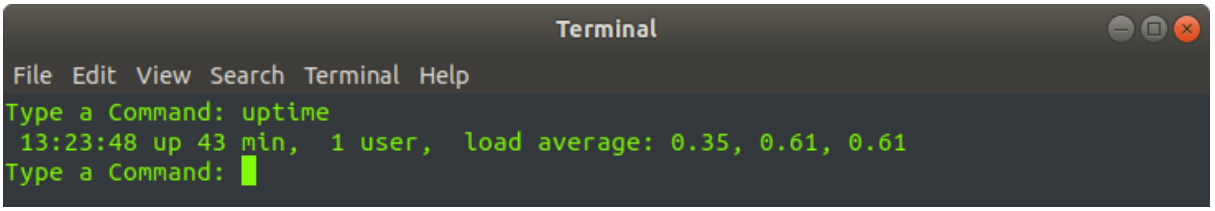


3. cal - This command displays a calendar.

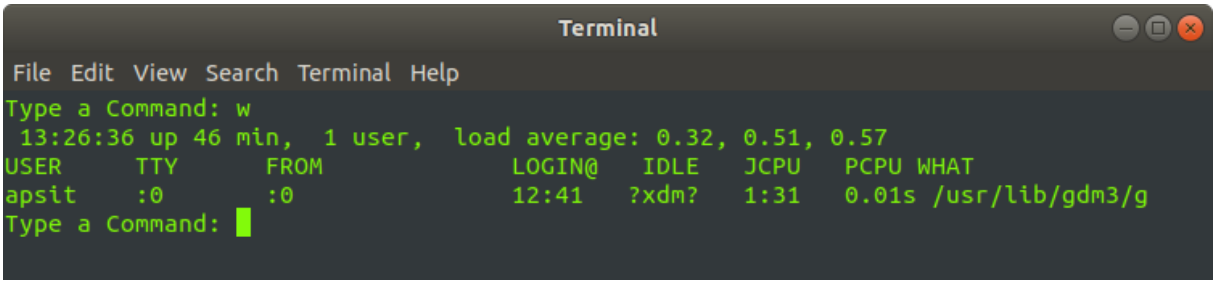




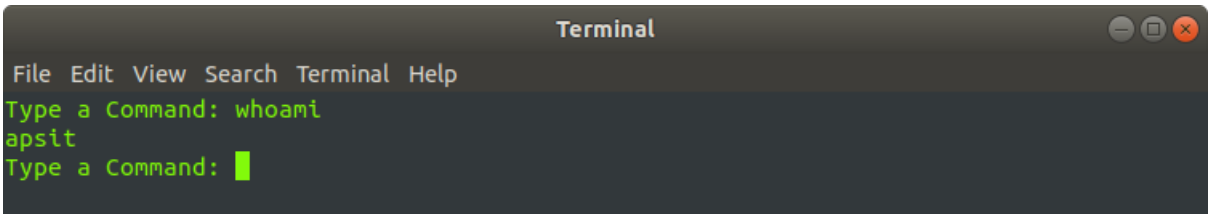
4. uptime - This command displays the no. of users and time duration that the system has been running.



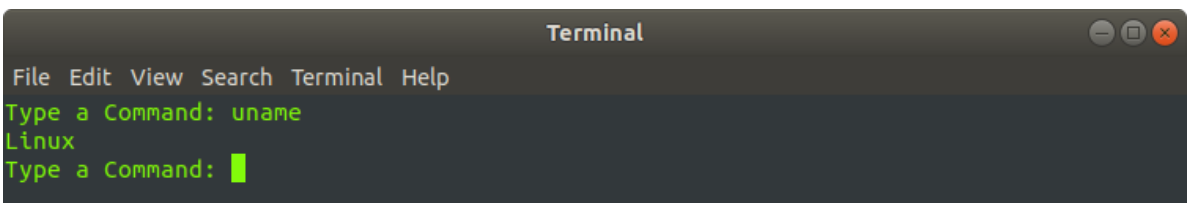
5. w - This command displays the system's uptime, user's name & login time, and what the user is doing.



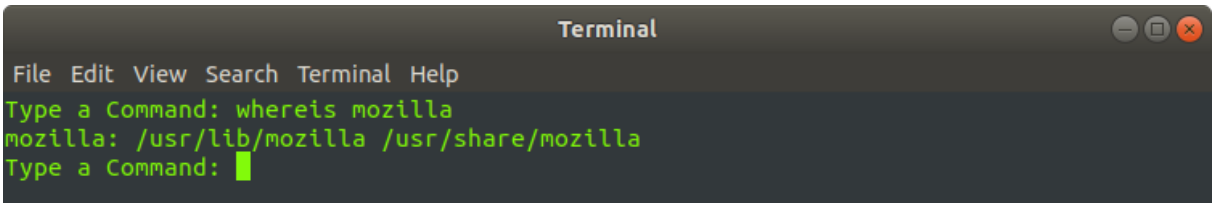
6. whoami - This command displays current user's name



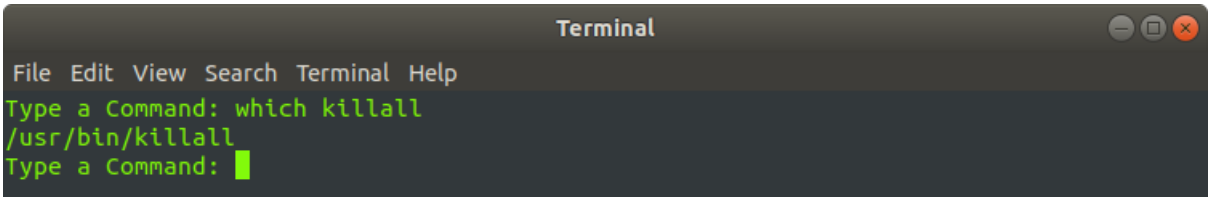
7. uname - This command displays the name of the current OS.



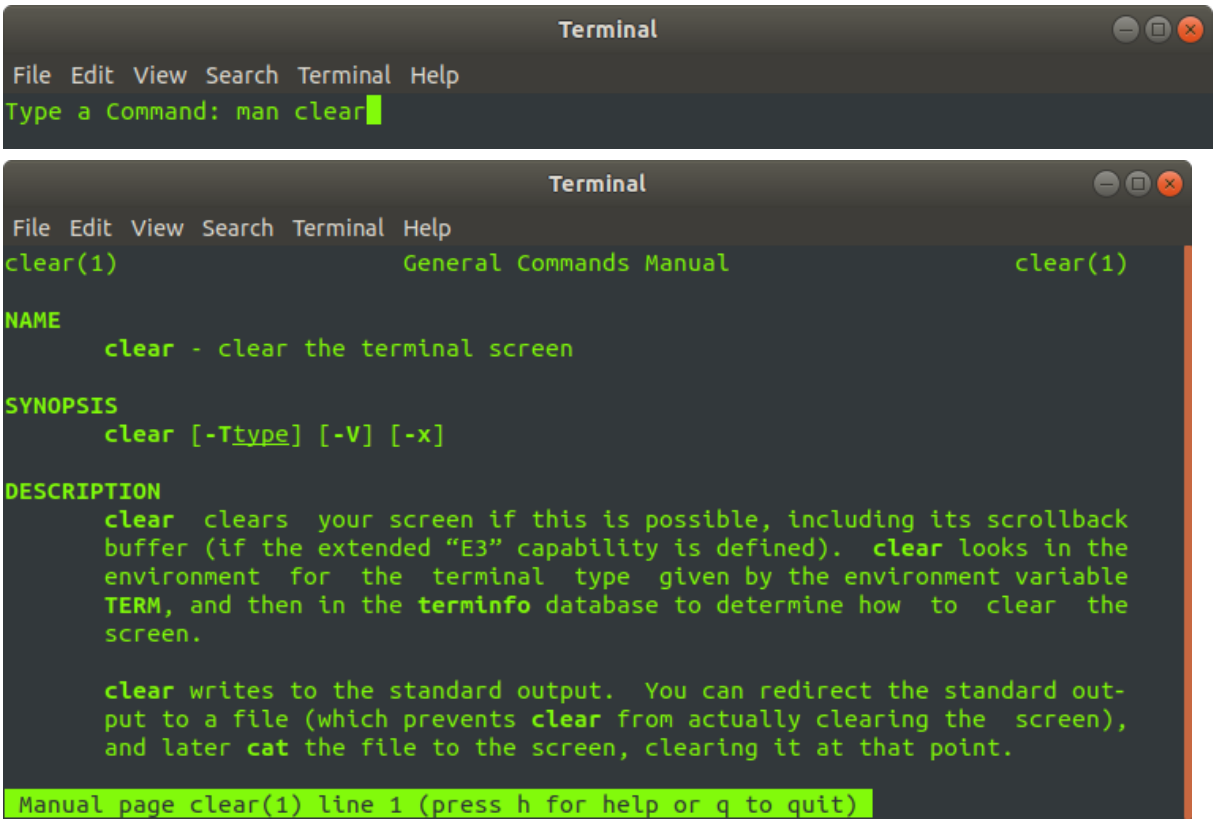
8. whereis - This command is used to find the path of an application



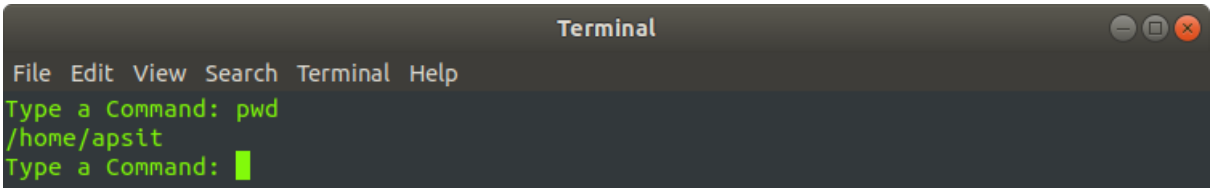
9. which - This command is used to find the path of an executable command.



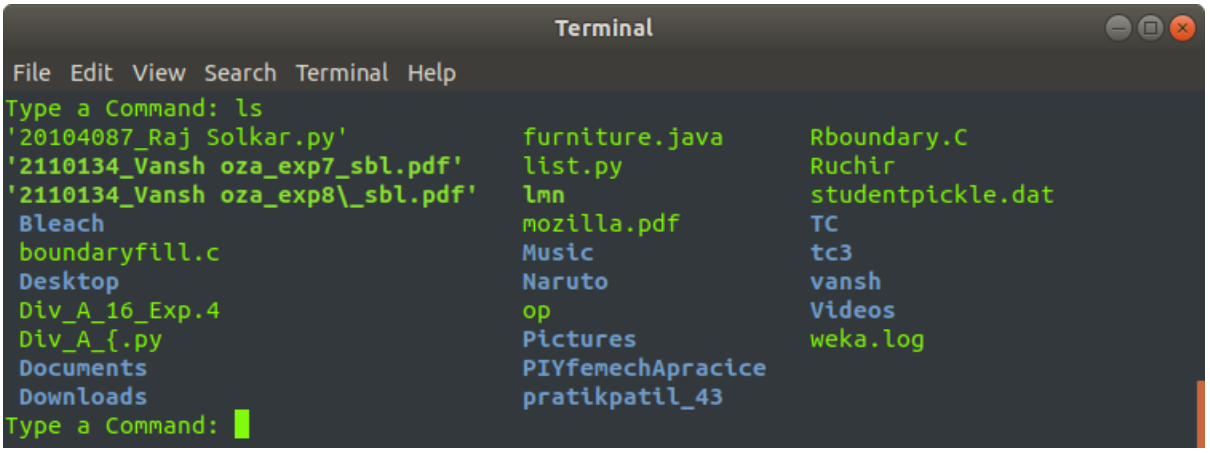
10. man - 'man' stands for manual. This command provides in-depth information about a given command.



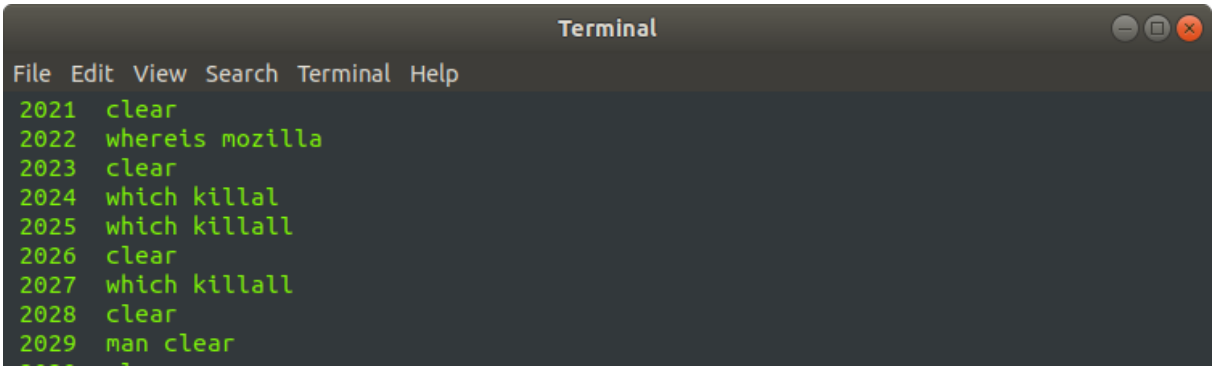
11. pwd - Stands for print working directory. This command displays the path of the working directory.



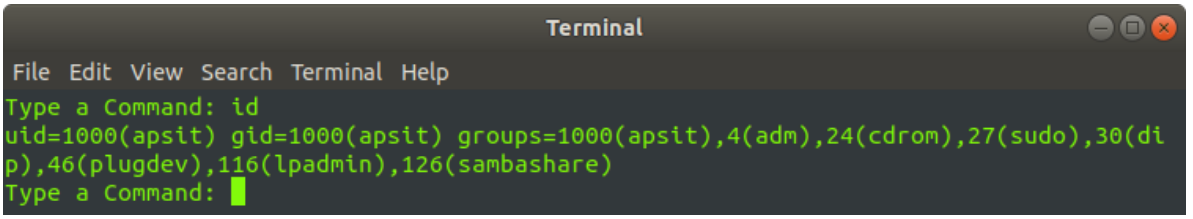
12. ls - This command displays a list of files and directories.



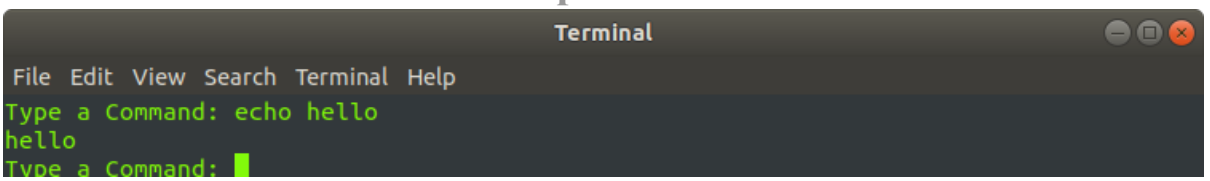
13. history - This command displays a history of the terminal commands



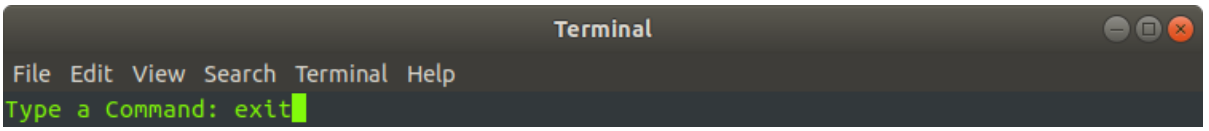
14. id - This command displays user ID and group ID.



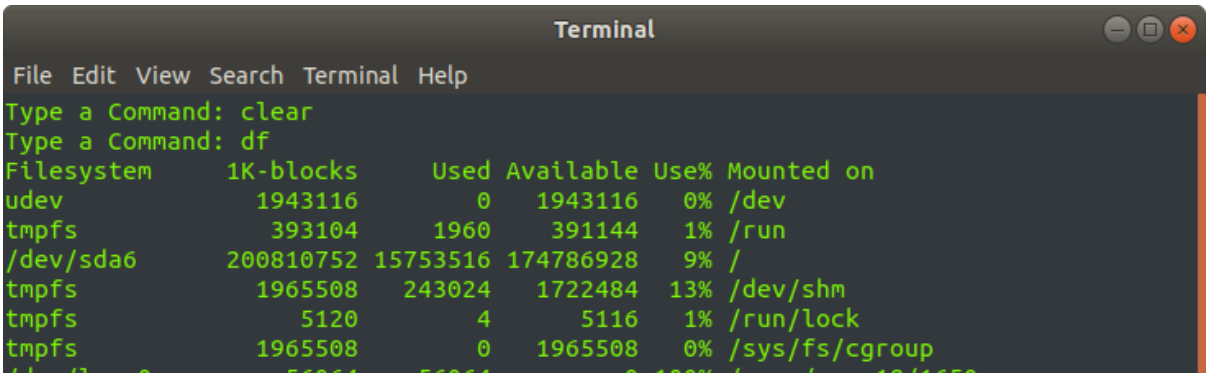
15. echo - This command returns the same text as specified.



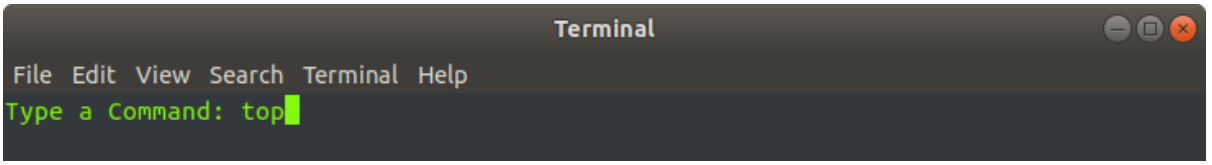
16. exit - This command will close your terminal.

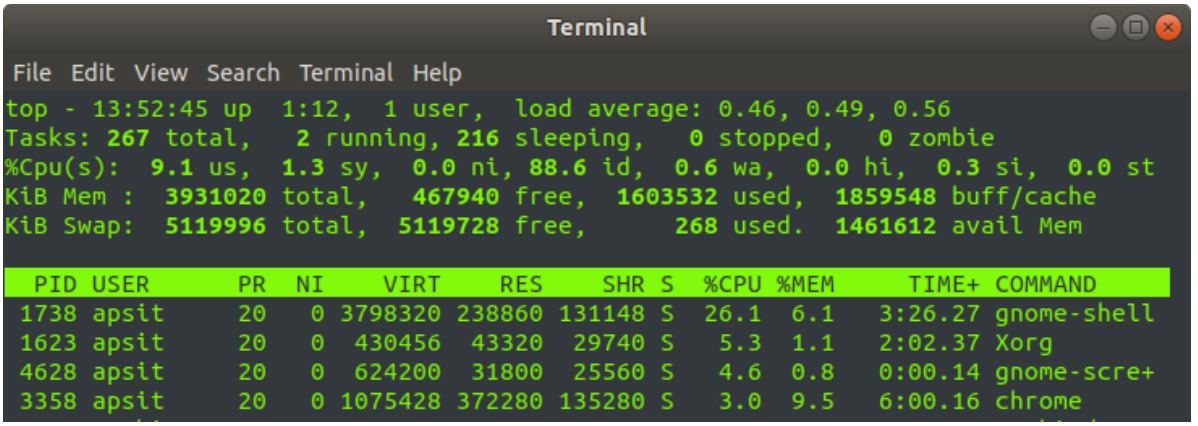


17. df - This command displays used and available disk space on the filesystem.



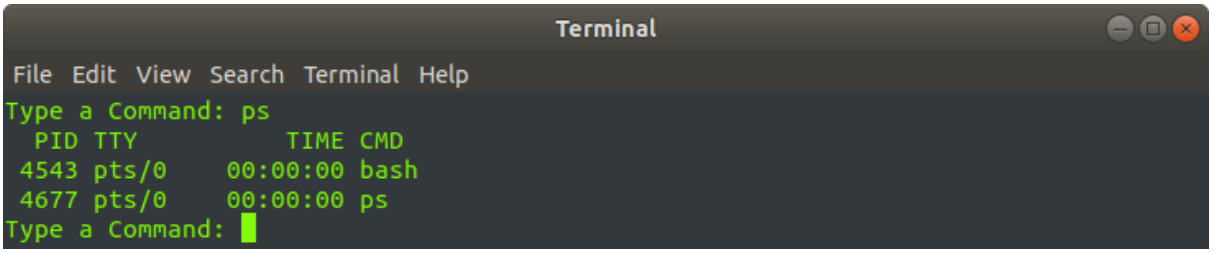
18. top - This command displays total, running, sleeping, stopped and zombie processes.

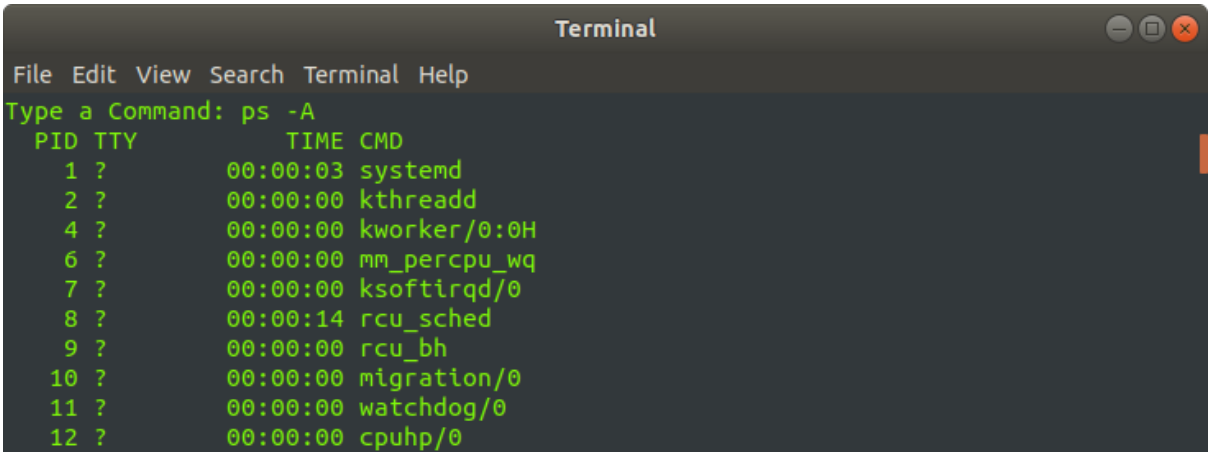


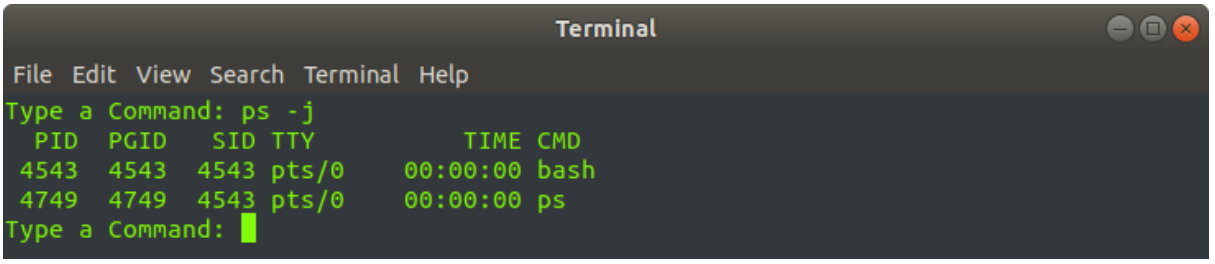


19. ps, ps -A, ps -j

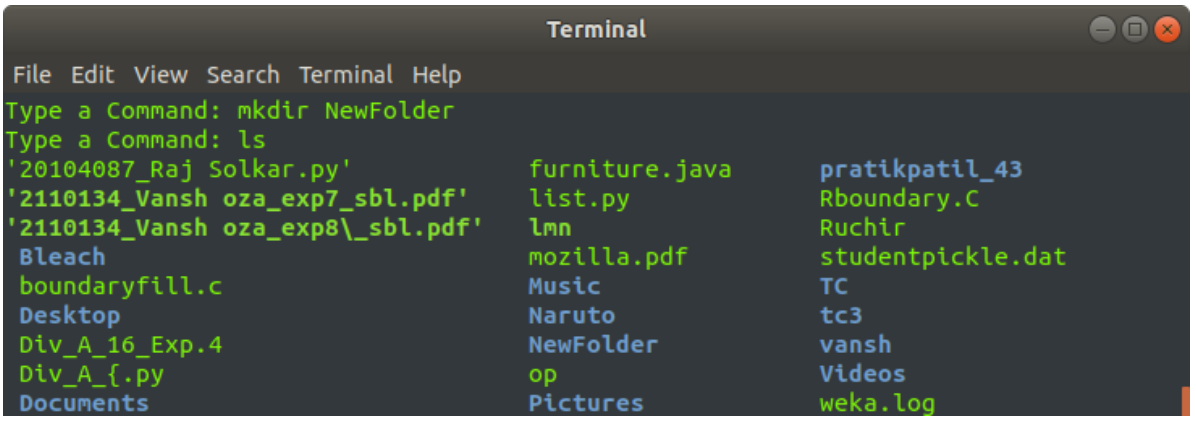
a. ps – Shows process status
b. ps -A – Displays information on all processes
c. ps -j – Displays process ID, process group ID and session ID



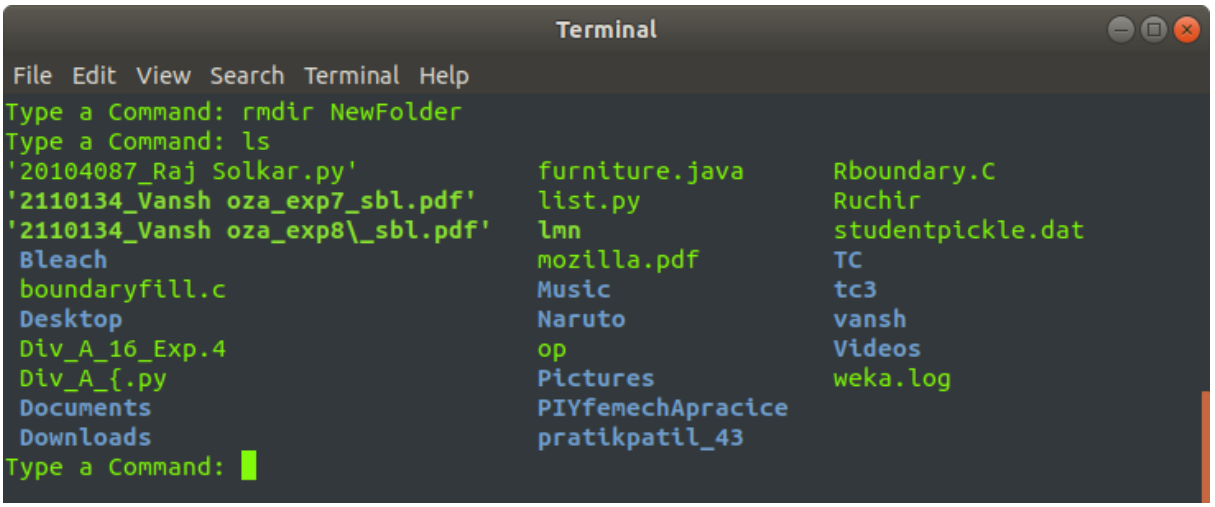




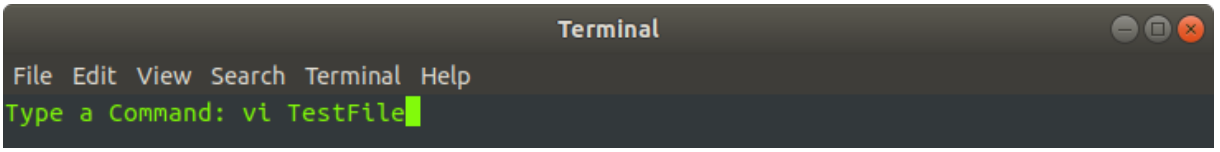
20. mkdir - This command creates a new directory

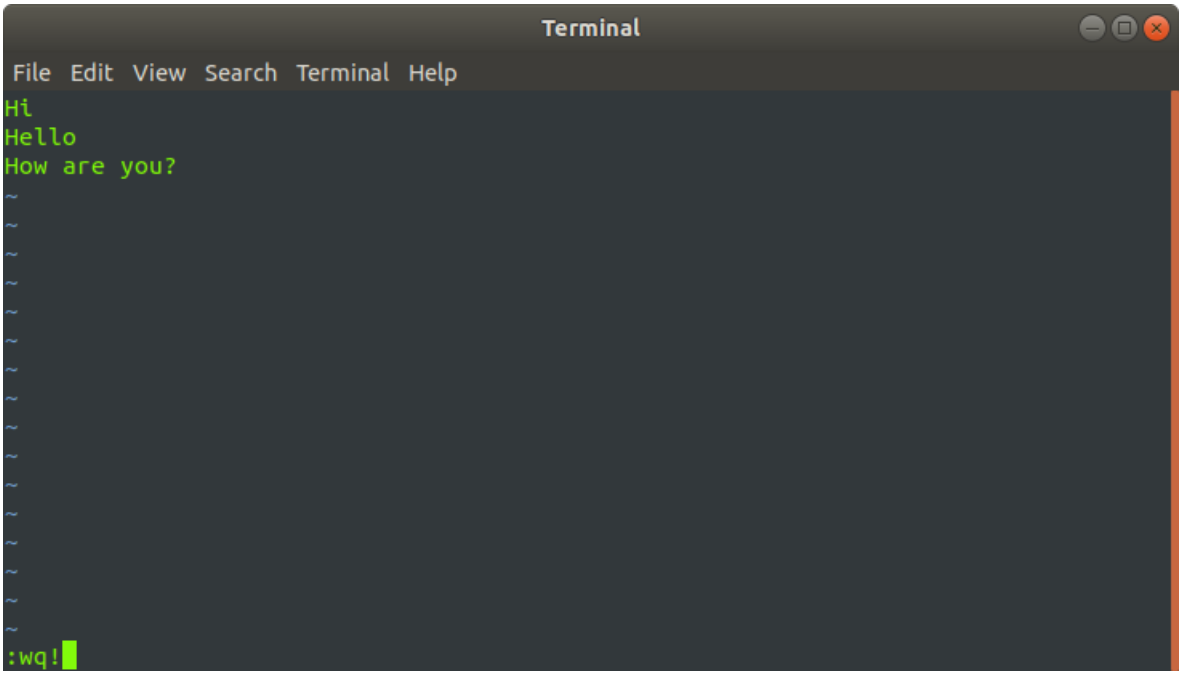


21. rmdir - This command removes an existing directory.

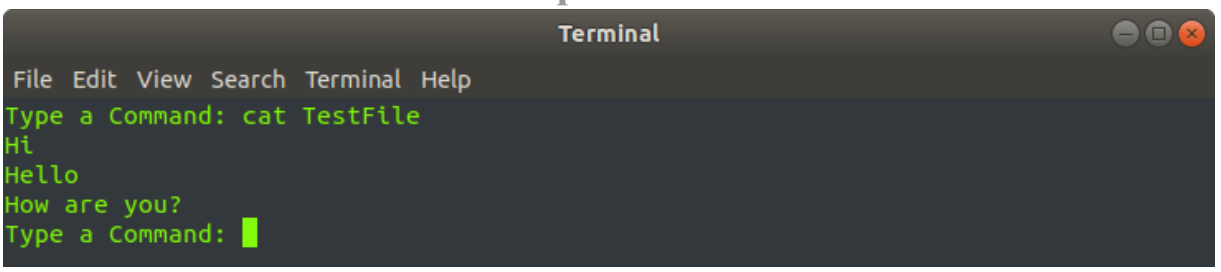


22. vi - A text editor used to create and edit text files.

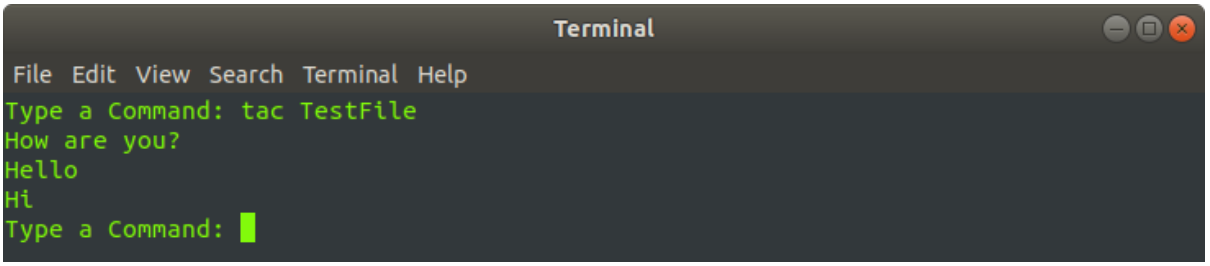




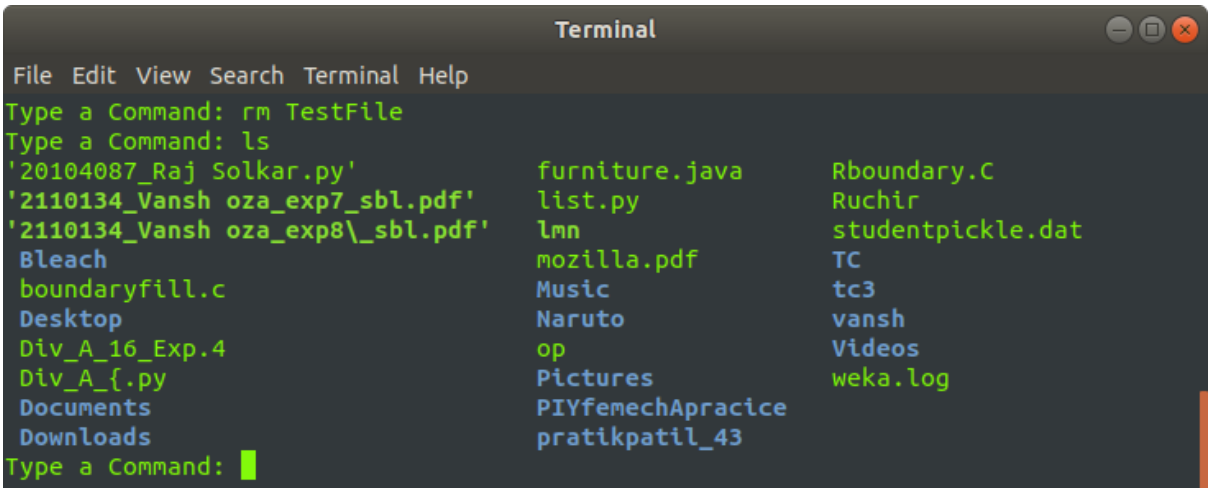
23. cat - This command displays the content of a specified file



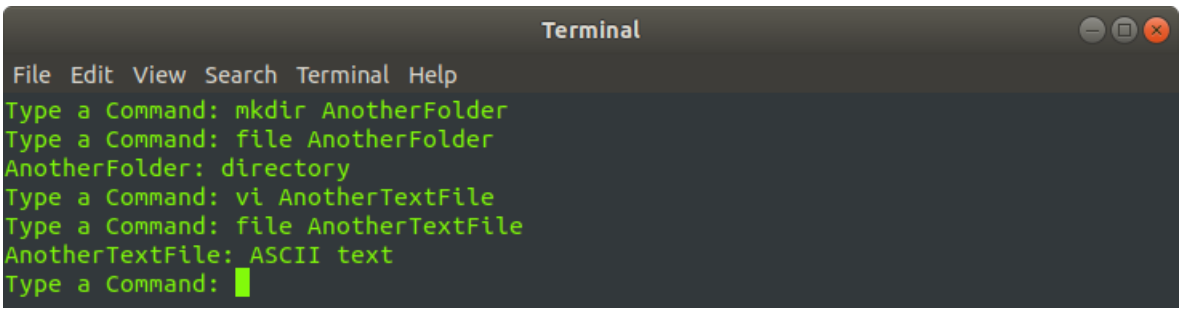
24. tac - This command displays the content of a specified file in the reverse order (i.e. last line first.)



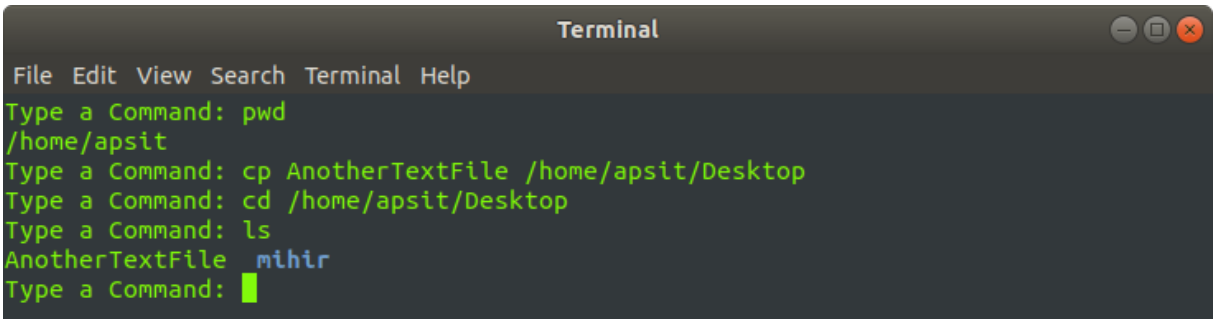
25. rm - This command is used to delete a file



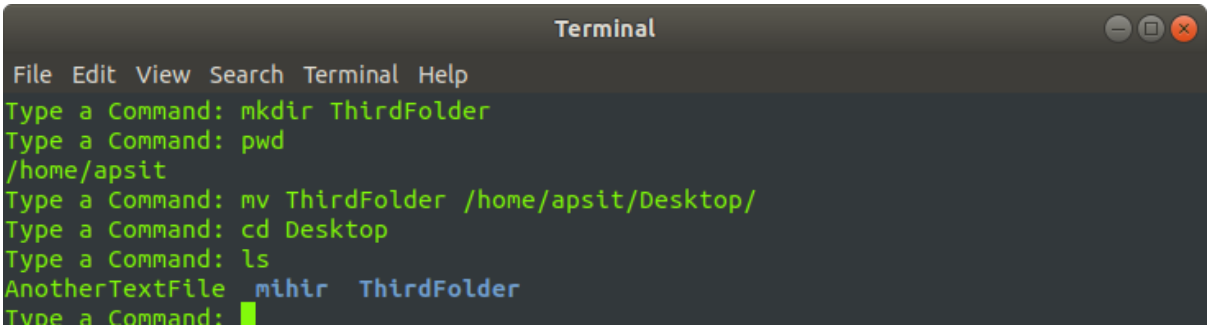
26. file - This command displays the file type of a specified file.



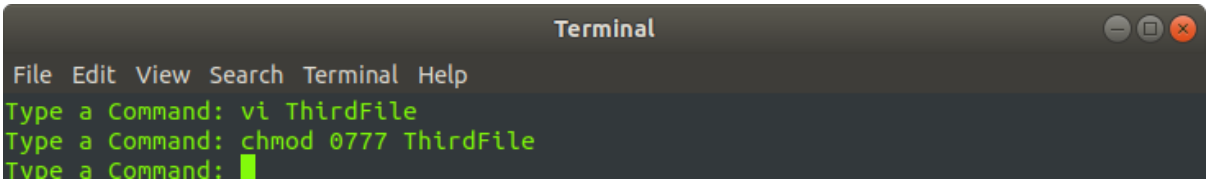
27. cp - This command is used to copy files and directories from one location to another.



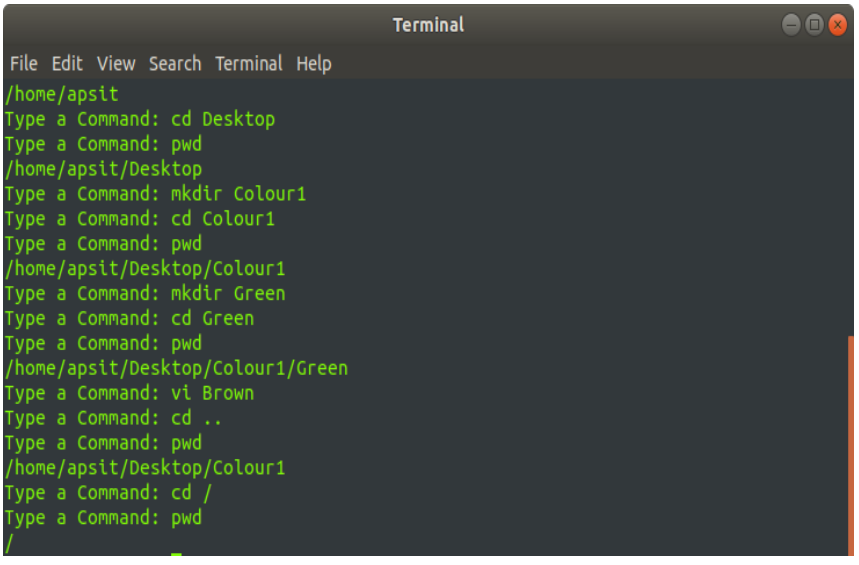
28. mv - This command is used to move files and directories from one location to another.



29. chmod - Stands for change mode. It is used to change access rights of a file.



cd - This command is used to change the current working directory.



EXP2 --Write a program to create processes in Linux using fork () system call.

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
int main()
{
fork();
printf("HELLO!\n");
return 0;
}
```

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
int main()
{
fork();
printf("HELLO!\n PID=%d\n",getpid());
return 0;
}
```

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
int main()
{
fork();
fork();
fork();
printf("hello\n");
return 0;
}
```

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
int main()
{
int pid=fork();
if(pid==0)
{
printf("I am child process having id %d\n",getpid());
}
else
{
printf("I am parent process having id %d\n",getppid());}
return 0;
}
```

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
int main()
{
int a;
a=fork();
if(a>0)
{
printf("hello\n");
}
else
{
printf("BIT\n");
}
return 0;
}

#include<stdio.h>
#include<unistd.h>
#include<sys/types.h>
int main()
{
pid_t p;
printf("before fork\n");
p=fork();
if(p==0)
{
printf("I am child having id %d\n",getpid());
printf("My parent's id is %d\n",getppid());
}
else{
printf("My child's id is %d\n",p);
printf("I am parent having id %d\n",getpid());
}
printf("Common\n");
}
```

==========

EXP3 ---Write a program to implement interprocess communication using Pipe()

//Q. Program to send a message from parent process to child process using pipe()

```
#include<stdio.h>
#include<unistd.h>
#include<sys/types.h>
#include<sys/wait.h>
int main()
{
int fd[2],n;
char buffer[100];
pid_t p;
pipe(fd); //creates a unidirectional pipe with two end fd[0] and fd[1]
p=fork();
```

```
if(p>0) //parent
{
printf("Parent Passing value to child\n");
write(fd[1],"hello\n",6); //fd[1] is the write end of the pipe
}
else // child
{
printf("Child printing received value\n");
n=read(fd[0],buffer,100); //fd[0] is the read end of the pipe
write(1,buffer,n);
}
}
```

==========

EXP 4--Write a program to demonstrate File handling and dup system calls in Linux

Program 1: Write a program using open() system call to read the first 10 characters of an existing file "test.txt" and print them on screen.

```
#include<unistd.h>
#include<sys/types.h>
#include<sys/stat.h>
#include<fcntl.h>
#include<stdio.h>
int main()
{
int n,fd;
char buff[50];
fd=open("test.txt",O_RDONLY); //opens test.txt in read mode and the file descriptor is saved in integer fd.
printf("The file descriptor of the file is: %d\n",fd); // the value of the file descriptor is printed.
n=read(fd,buff,10);//read 10 characters from the file pointed to by file descriptor fd and save them (buff)
write(1,buff,n); //write on the screen from the buffer
}
```

Output

Step1: create the file test.txt and write "1234567890abcdefghij54321" into it

```
$touch test.txt
```

Step2: compile the program

```
$gcc open.c
```

Step3: run

```
$./a.out
```

Program2: To read 10 characters from file "test.txt" and write them into non-existing file "towrite.txt"

```
#include<unistd.h>
#include<sys/types.h>
#include<sys/stat.h>
#include<fcntl.h>
int main()
{
int n,fd,fd1;
char buff[50];
fd=open("test.txt",O_RDONLY);
```

```
n=read(fd,buff,10);
fd1=open("towrite.txt",O_WRONLY|O_CREAT,0642);//use the pipe symbol (|) to separate
O_WRONLY and O_CREAT
write(fd1,buff,n);
}
```

Output : PS E:\S E M 4\os lab\CODES> cat test.txt
1234567890abcdefghij54321
PS E:\S E M 4\os lab\CODES> gcc 4b.c -o 4b
PS E:\S E M 4\os lab\CODES> ./4b
PS E:\S E M 4\os lab\CODES> cat towrite.txt
1234567890
PS E:\S E M 4\os lab\CODES>

Dup() system call ----→

Program 1: Program for dup() system call in C to duplicate a file descriptor

```
//dup.c
#include<unistd.h>
#include<stdio.h>
#include<fcntl.h>
int main()
{
int old_fd, new_fd;
old_fd=open("test.txt",O_RDWR);
printf("File descriptor is %d\n",old_fd);
new_fd=dup(old_fd);
printf("New file descriptor is %d\n",new_fd);
}
```

Program 2: Program to use dup2() system call in linux to duplicate a file descriptor.

```
#include<unistd.h>
#include<stdio.h>
#include<fcntl.h>
int main()
{
int old_fd, new_fd;
old_fd=open("test.txt",O_RDWR);
printf("File descriptor is %d\n",old_fd);
new_fd=dup2(old_fd,7);
printf("New file descriptor is %d\n",new_fd);
}
```

EXP 5 ---Write a program to implement interprocess communication using shared memory (SHM) in linux

Program 1: This program creates a shared memory segment, attaches itself to it and then writes some content into the shared memory segment.

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<sys/shm.h>
#include<string.h>
```

```
int main()
{
int i;
void *shared_memory;
char buff[100];
int shmid;
shmid=shmget((key_t)2345, 1024, 0666|IPC_CREAT);/*creates shared memory segment with
key 2345, having size 1024 bytes. IPC_CREAT is used to create the shared segment if it does
not exist. 0666 are the permisions on the shared segment*/
printf("Key of shared memory is %d\n",shmid);
shared_memory=shmat(shmid,NULL,0); //process attached to shared memory segment
printf("Process attached at %p\n",shared_memory); //this printsthe address where the segment is
attached with this process
printf("Enter some data to write to shared memory\n");
read(0,buff,100); //get some input from user
strcpy(shared_memory,buff); //data written to shared memory
printf("You wrote : %s\n",(char *)shared_memory);
}
```

Program2: This program attaches itself to the shared memory segment created in Program 1. Finally, it reads the content of the shared memory.

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<sys/shm.h>
#include<string.h>
int main()
{
int i;
void *shared_memory;
char buff[100];
int shmid;
shmid=shmget((key_t)2345, 1024, 0666);
printf("Key of shared memory is %d\n",shmid);
shared_memory=shmat(shmid,NULL,0); //process attached to shared memory segment
printf("Process attached at %p\n",shared_memory);
printf("Data read from shared memory is : %s\n",(char *)shared_memory);
}
```

EXP 6 ------→Write a program to create threads using Pthread in Linux.

Program 1: Program to create threads in linux. Thread prints 0-4 while the main process prints 20-24

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<pthread.h>
void *thread_function(void *arg);
int i,j;
int main() {
pthread_t a_thread; //thread declaration
pthread_create(&a_thread, NULL, thread_function, NULL);
//thread is created
```

```
pthread_join(a_thread, NULL); //process waits for thread to finish Comment this line to see the difference
printf("Inside Main Program\n");
for(j=20;j<25;j++)
{
printf("%d\n",j);
sleep(1);
}
}
void *thread_function(void *arg) {
// the work to be done by the thread is defined in this function
printf("Inside Thread\n");
for(i=0;i<5;i++)
{
printf("%d\n",i);
sleep(1);
}
}
```

For running code → $gcc Thread.c -lpthread

Program 2: Program to create a thread. The thread prints numbers from zero to n, where value of n is passed from the main process to the thread. The main process also waits for the thread to finish first and then prints from 20-24.

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<pthread.h>
#include<string.h>
void *thread_function(void *arg);
int i,n,j;
int main() {
char *m="5";
pthread_t a_thread; //thread declaration
void *result;
pthread_create(&a_thread, NULL, thread_function, m); //thread is created
pthread_join(a_thread, &result);
printf("Thread joined\n");
for(j=20;j<25;j++)
{
printf("%d\n",j);
sleep(1);
}
printf("thread returned %s\n",(char *)result);
}
void *thread_function(void *arg) {
n=atoi(arg);
for(i=0;i<n;i++)
{
printf("%d\n",i);
sleep(1);
}
pthread_exit("Done"); // Thread returns "Done"
}
```

Program 3: Program to create a thread to add two numbers

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<pthread.h>
#include<string.h>
void *thread_function(void *arg);
int num[2]={3,5};
int main() {
pthread_t a_thread; //thread declaration
void *result;
pthread_create(&a_thread, NULL, thread_function,(void *) num); //thread is created
pthread_join(a_thread, &result);
printf("Inside main process\n");
printf("Thread returned:%s\n",(char *)result);
}
void *thread_function(void *arg) {
printf("Inside thread\n");
int *x=arg;
int sum=x[0]+x[1];
printf("sum is %d\n",sum);
pthread_exit("sum calculated");
}
```

Exp 7 -Write a program to demonstrate Semaphores in Linux.

Program: Program creates two threads: one to increment the value of a shared variable and second to decrement the value of the shared variable. Both the threads make use of semaphore variable so that only one of the threads is executing in its critical section.

```
#include<pthread.h>
#include<stdio.h>
#include<semaphore.h>
#include<unistd.h>
void *fun1();
void *fun2();
int shared=1; //shared variable
sem_t s; //semaphore variable
int main()
{
sem_init(&s,0,1); /*initialize semaphore variable - 1st argument is address of variable, 2nd is
number of processes sharing semaphore, 3rd argument is the initial value of semaphore
variable*/

pthread_t thread1, thread2;
pthread_create(&thread1, NULL, fun1, NULL);
pthread_create(&thread2, NULL, fun2, NULL);
pthread_join(thread1, NULL);
pthread_join(thread2,NULL);
printf("Final value of shared is %d\n",shared); //prints the last updated value of sharedvariable
}
void *fun1()
{
int x;
```

```
sem_wait(&s); //executes wait operation on s
x=shared;//thread1 reads value of shared variable
printf("Thread1 reads the value as %d\n",x);
x++; //thread1 increments its value
printf("Local updation by Thread1: %d\n",x);
sleep(1); //thread1 is preempted by thread 2
shared=x; //thread one updates the value of shared variable
printf("Value of shared variable updated by Thread1 is: %d\n",shared);
sem_post(&s);
}
void *fun2()
{
int y;
sem_wait(&s);
y=shared;//thread2 reads value of shared variable
printf("Thread2 reads the value as %d\n",y);
y--; //thread2 increments its value
printf("Local updation by Thread2: %d\n",y);
sleep(1); //thread2 is preempted by thread 1
shared=y; //thread2 updates the value of shared variable
printf("Value of shared variable updated by Thread2 is: %d\n",shared);
sem_post(&s);
}
```

o/p--- gcc exp7.c -lpthread

---

exp 8 -- : Write a program to implement Basic Process Scheduling algorithms such as FCFS, SJF and RR.

Program1-FCFS

```
#include<stdio.h>
 #include<string.h>
 int main()
{
char pn[10][10],t[10];
int arr[10],bur[10],star[10],finish[10],tat[10],wt[10],i,j,n,temp; int totwt=0,tottat=0;
printf("Enter the number of processes:"); scanf("%d",&n);
for(i=0; i<n; i++)
{
printf("Enter the ProcessName, Arrival Time& Burst time:"); scanf("%s %d %d",&pn[i],&arr[i],&bur[i]);
}
for(i=0; i<n; i++)
{
for(j=0; j<n; j++)
{
if(arr[i]<arr[j])
{
temp=arr[i];
arr[i]=arr[j];
arr[j]=temp;
temp=bur[i];
bur[i]=bur[j];
bur[j]=temp;
strcpy(t,pn[i]);
strcpy(pn[i],pn[j]);
```

```
strcpy(pn[j],t);
}
}
}
for(i=0; i<n; i++)
{
if(i==0)
star[i]=arr[i]; else
star[i]=finish[i-1]; wt[i]=star[i]-arr[i]; finish[i]=star[i]+bur[i]; tat[i]=finish[i]-arr[i];
}
printf("\nPName Arrtime Burtime WaitTime Start TAT Finish"); for(i=0; i<n; i++)
{
 printf("\n%s\t%3d\t%3d\t%3d\t%3d\t%6d\t%6d",pn[i],arr[i],bur[i],wt[i],star[i],tat[i],finish[i]);
totwt+=wt[i]; tottat+=tat[i];
}
printf("\nAverage Waiting time:%f",(float)totwt/n);
printf("\nAverage Turn Around Time:%f",(float)tottat/n); return 0;
}
```

```
Enter the number of processes:3
Enter the ProcessName, Arrival Time& Burst Time:p1 0 5
Enter the ProcessName, Arrival Time& Burst Time:p2 1 3
Enter the ProcessName, Arrival Time& Burst Time:p3 2 4

PName Arrtime Burtime WaitTime Start TAT Finish
p1        0       5       0       0          5       5
p2        1       3       4       5          7       8
p3        2       4       6       8         10      12
Average Waiting time:3.333333
Average Turn Around Time:7.333333(base) apsit@apsit-HP-280-G2-MT-Legacy:
```

Program 2-SJF

```
#include <stdio.h>
int main()
{
int time, burst_time[10], at[10], sum_burst_time = 0, smallest, n, i;
int sumt = 0, sumw = 0;
printf("Enter the no of processes : ");
scanf("%d", & n);
for (i = 0; i < n; i++)
{
printf("Enter arrival time and burst time for process P%d : ", i + 1);
scanf("%d %d", & at[i], & burst_time[i]);
sum_burst_time += burst_time[i];
}
burst_time[9] = 9999;
printf("\nProcess\t\tTAT\t\tWaiting time\n");
for (time = 0; time < sum_burst_time;)
{
smallest = 9;
for (i = 0; i < n; i++)
{
if (at[i] <= time && burst_time[i] > 0 && burst_time[i] < burst_time[smallest])
smallest = i;
}
printf("P[%d]\t|\t%d\t|\t%d\n", smallest + 1, time + burst_time[smallest] - at[smallest], time -
at[smallest]);
```

```
sumt += time + burst_time[smallest] - at[smallest];
sumw += time - at[smallest];
time += burst_time[smallest];
 burst_time[smallest] = 0;
}
printf("\n\n average waiting time = %f", sumw * 1.0 / n);
printf("\n\n average turnaround time = %f\n", sumt * 1.0 / n);
return 0;
}
```

```
Enter the no of processes : 5
Enter arrival time and burst time for process P1 : 2 1
Enter arrival time and burst time for process P2 : 1 5
Enter arrival time and burst time for process P3 : 4 1
Enter arrival time and burst time for process P4 : 0 6
Enter arrival time and burst time for process P5 : 2 3

Process          TAT              Waiting time
P[4]     |       6        |       0
P[1]     |       5        |       4
P[3]     |       4        |       3
P[5]     |       9        |       6
P[2]     |       15       |       10


 average waiting time = 4.600000

 average turnaround time = 7.800000
```

Program 3-RR

```
#include<stdio.h>
struct process
{
   int id, AT, BT, WT, TAT;
};
struct process a[10];
// Declaration of the ready queue
int queue[100];
int front = -1;
int rear = -1;
// Function for inserting the element into the queue
void insert(int n)
{
   if (front == -1)
      front = 0;
   rear = rear + 1;
   queue[rear] = n;
}
// Function for deleting the element from the queue
int delete()
{
   int n;
   n = queue[front];
   front = front + 1;
   return n;
}
int main()
{
   int n, TQ, p, TIME = 0;
   int temp[10], exist[10] = {0};
   float total_wt = 0, total_tat = 0, Avg_WT, Avg_TAT;
   printf("Enter the number of processes\n");
   scanf("%d", &n);
```

```
 printf("Enter the arrival time and burst time of the processes\n");
 printf("AT BT\n");

 for(int i = 0; i < n; i++)
 {
    scanf("%d%d", &a[i].AT, &a[i].BT);
    a[i].id = i;
    temp[i] = a[i].BT;
 }
printf("Enter the time quantum\n");
 scanf("%d", &TQ);

 // Logic for round robin scheduling
 // Insert first process into ready queue
 // insert(0);
 // exist[0] = 1;
 // Until ready queue is empty
 // while(front <= rear)
 {
    p = delete();
    if(a[p].BT >= TQ)
    {
       a[p].BT = a[p].BT - TQ;
       TIME = TIME + TQ;
    }
    else
    {
       TIME = TIME + a[p].BT;
       a[p].BT = 0;
    }

    // If process is not exist in the ready queue
    // even a single time then insert it
    // if it arrives at time 'TIME'
    for(int i = 0; i < n; i++)
    {
       if(exist[i] == 0 && a[i].AT <= TIME)
       {
          insert(i);
          exist[i] = 1;
       }
    }
// If process is completed
    // if(a[p].BT == 0)
    {
       a[p].TAT = TIME - a[p].AT;
       a[p].WT = a[p].TAT - temp[p];
       total_tat = total_tat + a[p].TAT;
       total_wt = total_wt + a[p].WT;
    }
    // else
    {
       insert(p);
```

```
    }
  }
  Avg_TAT = total_tat / n;
  Avg_WT = total_wt / n;
  printf("ID WT TAT\n");
  for(int i = 0; i < n; i++)
  {
    printf("%d %d %d\n", a[i].id, a[i].WT, a[i].TAT);
  }
  printf("Average waiting time of the processes is: %f\n", Avg_WT);
  printf("Average turnaround time of the processes is: %f\n", Avg_TAT);
  return 0;
}
```

Exp 9 - Aim: Write a program to demonstrate paging.

```
#include<stdio.h>
void main()
{
int memsize=32;
int pagesize, nofpage;
int p[100];
int frameno, offset;
int logadd, phyadd;
int i;
int choice=0;
printf("\nYour memsize is %d ",memsize);
printf("\nEnter page size:");
scanf("%d",&pagesize);
nofpage=memsize/pagesize;
for(i=0;i<nofpage;i++)
{
printf("\nEnter the frame of page%d:",i);
scanf("%d",&p[i]);
}
do
{
printf("\nEnter a logical address:");
scanf("%d",&logadd);
frameno=logadd/pagesize;
offset=logadd%pagesize;
phyadd=(p[frameno]*pagesize)+offset;
printf("\nPhysical address is:%d",phyadd);
printf("\nDo you want to continue(1/0)?:");
scanf("%d",&choice);
}while(choice==1);
}
```

Op-→

Your memsize is 32 Enter page size:4 Enter the frame of page0:5 Enter the frame of page1:6 Enter the frame of page2:4 Enter the frame of page3:2 Enter the frame of page4:7 Enter the frame of page5:8 Enter the frame of page6:3 Enter the frame of page7:2 Enter a logical address:4 Physical address is:24 Do you want to continue(1/0)?

Exp 10 Write a program to demonstrate Disk scheduling algorithms like FCFS, SSTF, SCAN and LOOK

Program 1: FCFS(First Come First Served Scheduling:)

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
int RQ[100],i,n,TotalHeadMoment=0,initial;
printf("Enter the number of Requests\n");
scanf("%d",&n);
printf("Enter the Requests sequence\n");
for(i=0;i<n;i++)
scanf("%d",&RQ[i]);
printf("Enter initial head position\n");
scanf("%d",&initial);
// logic for FCFS disk scheduling
for(i=0;i<n;i++)
{
TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);
initial=RQ[i];
}
printf("Total head moment is %d",TotalHeadMoment);
return 0;
}
```

Op➔

```
Enter the number of Requests
8
Enter the Requests sequence
95 180 34 119 11 123 62 64
Enter initial head position
50
Total head moment is 644
```

Program 2. Shortest Seek Time First(SSTF)Scheduling:

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
int RQ[100],i,n,TotalHeadMoment=0,initial,count=0;
printf("Enter the number of Requests\n");
scanf("%d",&n);
printf("Enter the Requests sequence\n");
for(i=0;i<n;i++)
scanf("%d",&RQ[i]);
printf("Enter initial head position\n");
scanf("%d",&initial);// logic for sstf disk scheduling
/* loop will execute until all process is completed*/
while(count!=n)
{
   int min=1000,d,index;
for(i=0;i<n;i++)
{
```

```
d=abs(RQ[i]-initial);
if(min>d)
{
min=d;
index=i;
}}
TotalHeadMoment=TotalHeadMoment+min;
initial=RQ[index];// 1000 is for max// you can use any number
RQ[index]=1000;
count++;
}
printf("Total head movement is %d",TotalHeadMoment);
return 0;
}
```

Program 3: SCAN Scheduling:

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
int RQ[100],i,j,n,TotalHeadMoment=0,initial,size,move;
printf("Enter the number of Requests\n");
scanf("%d",&n);
printf("Enter the Requests sequence\n");
for(i=0;i<n;i++)
scanf("%d",&RQ[i]);
printf("Enter initial head position\n");
scanf("%d",&initial);
printf("Enter total disk size\n");
scanf("%d",&size);
printf("Enter the head movement direction for high 1 and for low 0\n");
scanf("%d",&move);
// logic for Scan disk scheduling
/*logic for sort the request array */
for(i=0;i<n;i++)
{
for(j=0;j<n-i-1;j++)
{
if(RQ[j]>RQ[j+1])
{
int temp;
temp=RQ[j];
RQ[j]=RQ[j+1];
RQ[j+1]=temp;
}}}
int index;
for(i=0;i<n;i++)
{
if(initial<RQ[i])
{
index=i;
break;
}
}// if movement is towards high value
```

```
if(move==1)
{
   for(i=index;i<n;i++)
{
TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);
initial=RQ[i];
}// last movement for max size
TotalHeadMoment=TotalHeadMoment+abs(size-RQ[i-1]-1);
initial = size-1;
for(i=index-1;i>=0;i--)
{
TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);
initial=RQ[i];
}
}// if movement is towards low value
else
{
for(i=index-1;i>=0;i--)
{
TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);
initial=RQ[i];
}// last movement for min size
TotalHeadMoment=TotalHeadMoment+abs(RQ[i+1]-0);
initial =0;
for(i=index;i<n;i++)
{
TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);
initial=RQ[i];
}}
printf("Total head movement is %d",TotalHeadMoment);
return 0;
}
```

Program 4. LOOK Scheduling:

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
int RQ[100],i,j,n,TotalHeadMoment=0,initial,size,move;
printf("Enter the number of Requests\n");
scanf("%d",&n);
printf("Enter the Requests sequence\n");
for(i=0;i<n;i++)
scanf("%d",&RQ[i]);
printf("Enter initial head position\n");
scanf("%d",&initial);
printf("Enter total disk size\n");
scanf("%d",&size);
printf("Enter the head movement direction for high 1 and for low 0\n");
scanf("%d",&move);
// logic for look disk scheduling
/*logic for sort the request array */
for(i=0;i<n;i++)
{
```

```
for(j=0;j<n-i-1;j++)
{
if(RQ[j]>RQ[j+1])
{
int temp;
temp=RQ[j];
RQ[j]=RQ[j+1];
RQ[j+1]=temp;
}
}
}
int index;
for(i=0;i<n;i++)
{
if(initial<RQ[i])
{
index=i;
break;
}
}
// if movement is towards high value
if(move==1)
{
   for(i=index;i<n;i++)
{
TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);
initial=RQ[i];
}
for(i=index-1;i>=0;i--)
{
TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);
initial=RQ[i];
}
}
// if movement is towards low value
else
{
for(i=index-1;i>=0;i--)
{
TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);
initial=RQ[i];
}
for(i=index;i<n;i++)
{
TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);
initial=RQ[i];
}
}
printf("Total head movement is %d",TotalHeadMoment);
return 0;
}
```

Exp11-→ Aim: Write a program to demonstrate file allocation method such as Contiguous allocation.

```
#include<stdio.h>
void main()
{
int i,j,n,block[20],start;
printf("Enter the no. of file:\n");
scanf("%d",&n);
printf("Enter the number of blocks needed foreach file:\n");
for(i=0;i<n;i++)
scanf("%d",&block[i]);
start=0;
printf("\t\tFile name\tStart\tSize of file\t\t\n");
printf("\n\t\tFile1\t\t%d\t\t\t%d\n",start,block[0]);
for(i=2;i<=n;i++)
{
start=start+block[i-2];
printf("\t\tFile%d \t\t%d\t\t%d\n",i,start,block[i-1]);
}
}
```

Op->

```
Enter the no. of file:
4
Enter the number of blocks needed foreach file:
3
5
6
1
                File name       Start   Size of file

                File1           0                       3
                File2           3               5
                File3           8               6
                File4           14              1
```