17BCE0581
SATYAM SINGH CHAUHAN

OPERATING SYSTEMS
CSE2005

IPC Through Shared Memory

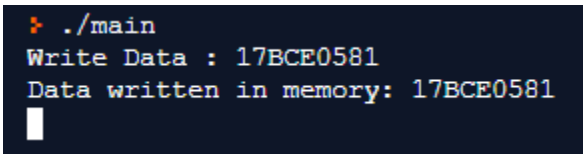
AIM:

Inter Process Communication through shared memory where two or more process can access the common memory. And communication is done via this shared memory where changes made by one process can be viewed by another process.

CODE For Writing in Memory

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

int main()
{
    key_t key = ftok("shmfile",65);
    int shmid = shmget(key,1024,0666|IPC_CREAT);
    char *str = (char*) shmat(shmid,(void*)0,0);
    printf("Write Data : ");
    gets(str);
    printf("Data written in memory: %s\n",str);
    shmdt(str);
    return 0;
}
```

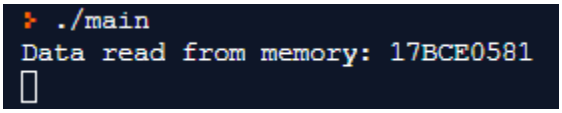
OUTPUT



CODE For Accessing from Memory

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

int main()
{
    key_t key = ftok("shmfile",65);
    int shmid = shmget(key,1024,0666|IPC_CREAT);
    char *str = (char*) shmat(shmid,(void*)0,0);
    printf("Data read from memory: %s\n",str);
    shmdt(str);
    shmctl(shmid,IPC_RMID,NULL);
    return 0;
}
```

OUTPUT

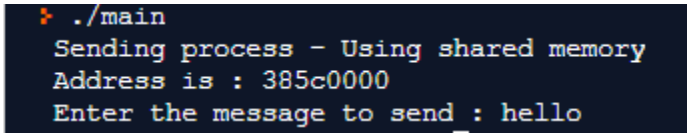


PROGRAM TO ACCESS SHARED MEMORY

CODE FOR SENDING THE MESSAGE

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

#define SHMKEY 75
#define K 1024

main()
{
   int shmid;
   char *addr1;

   printf(" Sending process - Using shared memory");
   shmid= shmget(SHMKEY, 128*K, 0777|IPC_CREAT);
   addr1= shmat(shmid,0,0);

   printf("\n Address is : %x",addr1);
   printf("\n Enter the message to send : ");
   scanf("%s", addr1);
}
```

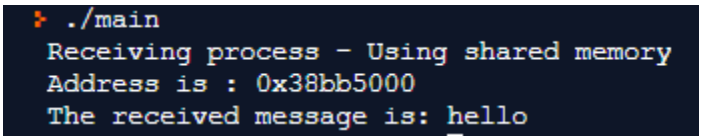
OUTPUT



CODE FOR RECEIVING THE MESSAGE

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

#define SHMKEY 75
#define K 1024

main()
{
   int shmid;
   char *addr1;

   printf(" Receiving process - Using shared memory");
   shmid= shmget(SHMKEY, 128*K, 0777|IPC_CREAT);
   addr1= shmat(shmid,0,0);

   printf("\n Address is : 0x%x",addr1);
   printf("\n The received message is: %s", addr1);
   printf("\n\n");
}
```

OUTPUT



SCHEDULING ALGORITHMS

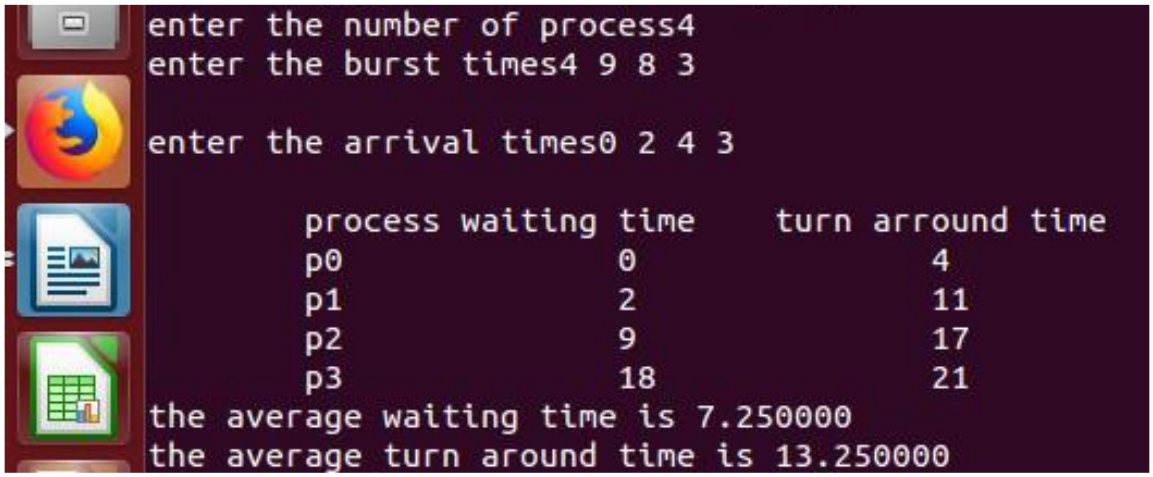
AIM

Given n processes with their burst times, the task is to find average waiting time and average turn around time using FCFS scheduling algorithm.

CODE FOR PERFORMING FCFS

```
#include<stdio.h>
main()
{
      int n,a[10],b[10],t[10],w[10],g[10],i,m;
      float att=0,awt=0;
      for(i=0;i<10;i++)
      {
      a[i]=0; b[i]=0; w[i]=0; g[i]=0;
      }
      printf("enter the number of process");
      scanf("%d",&n);
      printf("enter the burst times");
      for(i=0;i<n;i++)
      scanf("%d",&b[i]);
      printf("\nenter the arrival times");
      for(i=0;i<n;i++)
      scanf("%d",&a[i]);
      g[0]=0;
      for(i=0;i<10;i++)
      g[i+1]=g[i]+b[i];
      for(i=0;i<n;i++)
      {
      w[i]=g[i]-a[i];
      t[i]=g[i+1]-a[i];
      awt=awt+w[i];
      att=att+t[i];
      }
      awt =awt/n;
      att=att/n;
      printf("\n\tprocess\twaiting time\tturn arround time\n");
      for(i=0;i<n;i++)
      {
      printf("\tp%d\t\t%d\t\t%d\n",i,w[i],t[i]);
      }
      printf("the average waiting time is %f\n",awt);
      printf("the average turn around time is %f\n",att);
}
```

OUTPUT



AIM

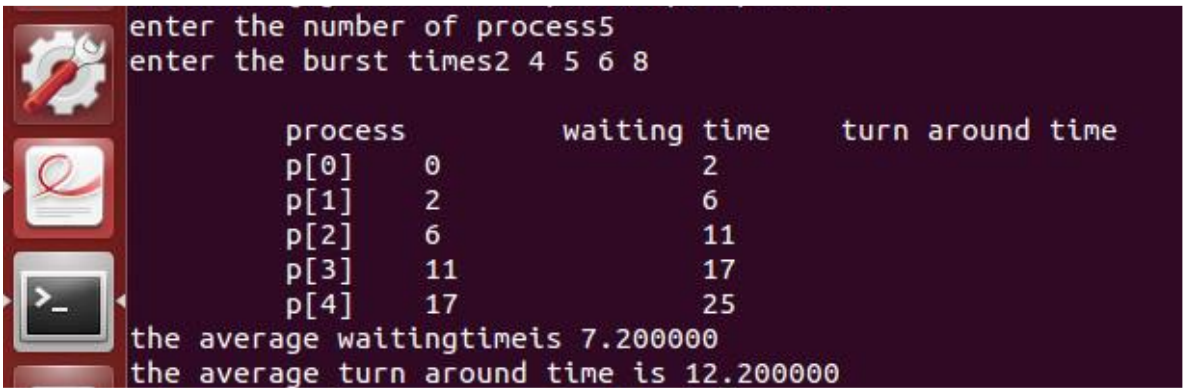
Given n processes with their burst times, the task is to find average waiting time and average turn around time using SJF scheduling algorithm.

CODE FOR PERFORMING SJF

```
#include<stdio.h>
int main()
{
	int n,j,temp,temp1,temp2,pr[10],b[10],t[10],w[10],p[10],i;
	float att=0,awt=0;
	for(i=0;i<10;i++)
	{
		b[i]=0;w[i]=0;
	}
	printf("enter the number of process");
	scanf("%d",&n);
	printf("enter the burst times");
	for(i=0;i<n;i++)
	{
		scanf("%d",&b[i]);
		p[i]=i;
	}
	for(i=0;i<n;i++)
	{
	for(j=i;j<n;j++)
	{
	if(b[i]>b[j])
	{
```

```
        temp=b[i];
        temp1=p[i];
        b[i]=b[j];
        p[i]=p[j];
        b[j]=temp;
        p[j]=temp1;
        }
        }
        }
        w[0]=0;
        for(i=0;i<n;i++)
        w[i+1]=w[i]+b[i];
        for(i=0;i<n;i++)
        {
        t[i]=w[i]+b[i];
        awt=awt+w[i];
        att=att+t[i];
        }
        awt=awt/n;
        att=att/n;
        printf("\n\t process \t waiting time \t turn around time \n");
        for(i=0;i<n;i++)
        printf("\t p[%d] \t\t\t\t %d \t\t\t\t %d \n",p[i],w[i],t[i]);
        printf("the average waitingtimeis %f\n",awt);
        printf("the average turn around time is %f\n",att);
        return 1;
}
```

OUTPUT



AIM

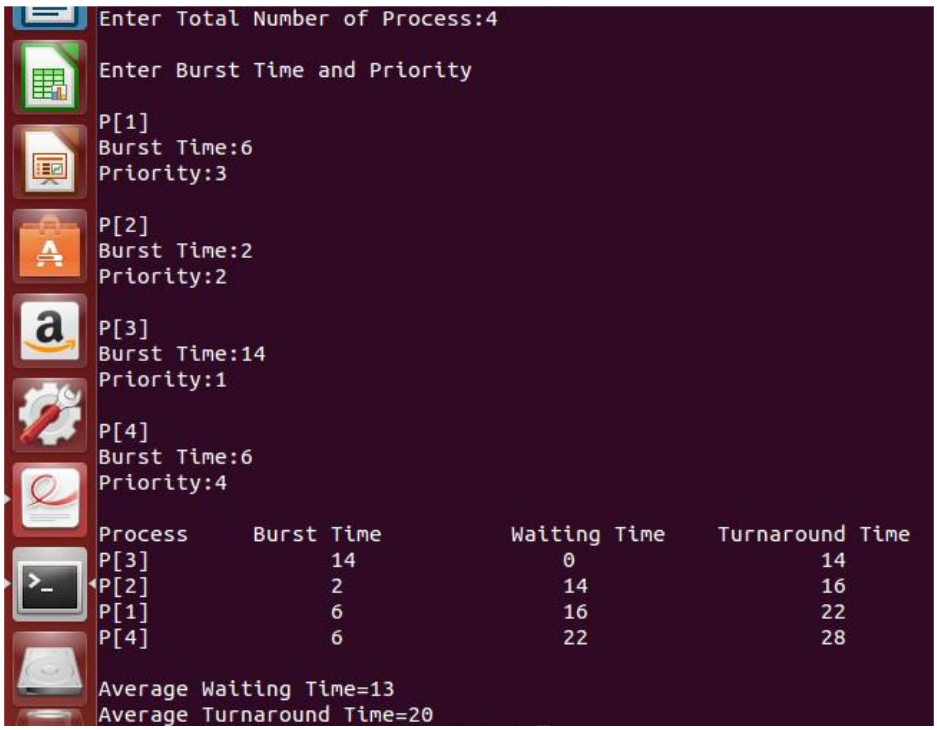
Given n processes with their burst times and priorities, the task is to find average waiting time and average turn around time using PRIORITY scheduling algorithm.

CODE FOR PERFORMING PRIOIRTY

```
#include<stdio.h>
int main()
{
      int
      bt[20],p[20],wt[20],tat[20],pr[20],i,j,n,total=0,pos,temp,avg_wt,avg_ta
      t;
      printf("Enter Total Number of Process:");
      scanf("%d",&n);
      printf("\nEnter Burst Time and Priority\n");
      for(i=0;i<n;i++)
      {
      printf("\nP[%d]\n",i+1);
      printf("Burst Time:");
      scanf("%d",&bt[i]);
      printf("Priority:");
      scanf("%d",&pr[i]);
      p[i]=i+1; //contains process number
      }
      //sorting burst time, priority and process number in ascending order
      using selection sort
      for(i=0;i<n;i++)
      {
      pos=i;
      for(j=i+1;j<n;j++)
      {
      if(pr[j]<pr[pos])
      pos=j;
      }
      temp=pr[i];
      pr[i]=pr[pos];
      pr[pos]=temp;
      temp=bt[i];
      bt[i]=bt[pos];
      bt[pos]=temp;
      temp=p[i];
      p[i]=p[pos];
      p[pos]=temp;
      }
      wt[0]=0; //waiting time for first process is zero
```

```
        //calculate waiting time
        for(i=1;i<n;i++)
        {
        wt[i]=0;
        for(j=0;j<i;j++)
        wt[i]+=bt[j];
        total+=wt[i];
        }
        avg_wt=total/n; //average waiting time
        total=0;
        printf("\nProcess\t Burst Time \tWaiting Time\tTurnaround Time");
        for(i=0;i<n;i++)
        {
        tat[i]=bt[i]+wt[i]; //calculate turnaround time
        total+=tat[i];
        printf("\nP[%d]\t\t %d\t\t\t\t %d\t\t\t\t%d",p[i],bt[i],wt[i],tat[i]);
        }
        avg_tat=total/n; //average turnaround time
        printf("\n\nAverage Waiting Time=%d",avg_wt);
        printf("\nAverage Turnaround Time=%d\n",avg_tat);
        return 0;
}
```

OUTPUT



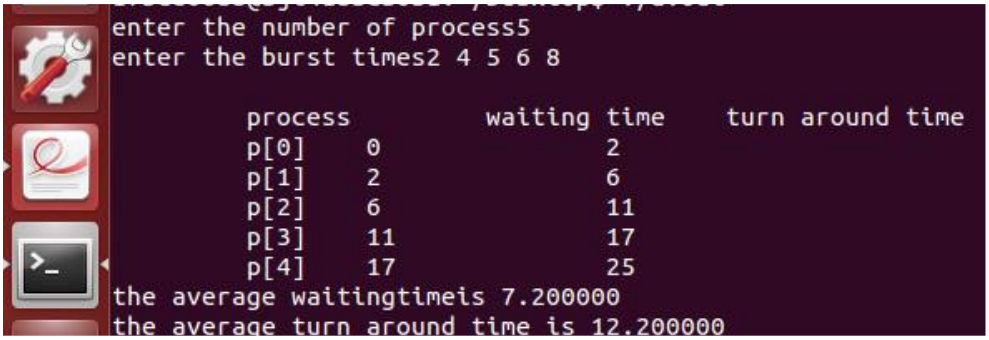
AIM

Given n processes with their burst times, the task is to find average waiting time and average turn around time using SRTF scheduling algorithm.

CODE FOR PERFORMING SRTF

```
#include <stdio.h>
int main()
{
      int a[10],b[10],x[10],i,j,smallest,count=0,time,n;
      double avg=0,tt=0,end;
      printf("enter the number of Processes:\n");
      scanf("%d",&n);
      printf("enter arrival time\n");
      for(i=0;i<n;i++)
      scanf("%d",&a[i]);
      printf("enter burst time\n");
      for(i=0;i<n;i++)
      scanf("%d",&b[i]);
      for(i=0;i<n;i++)
      x[i]=b[i];
      b[9]=9999;
      for(time=0;count!=n;time++)
      {
      smallest=9;
      for(i=0;i<n;i++)
      {
      if(a[i]<=time && b[i]<b[smallest] && b[i]>0 )
      smallest=i;
      }
      b[smallest]--;
      if(b[smallest]==0)
      {
      count++;
      end=time+1;
      avg=avg+end-a[smallest]-x[smallest];
      tt= tt+end-a[smallest];
      }
      }
      printf("\n\nAverage waiting time = %lf\n",avg/n);
      printf("Average Turnaround time = %lf",tt/n);
      return 0;
}
```

OUTPUT



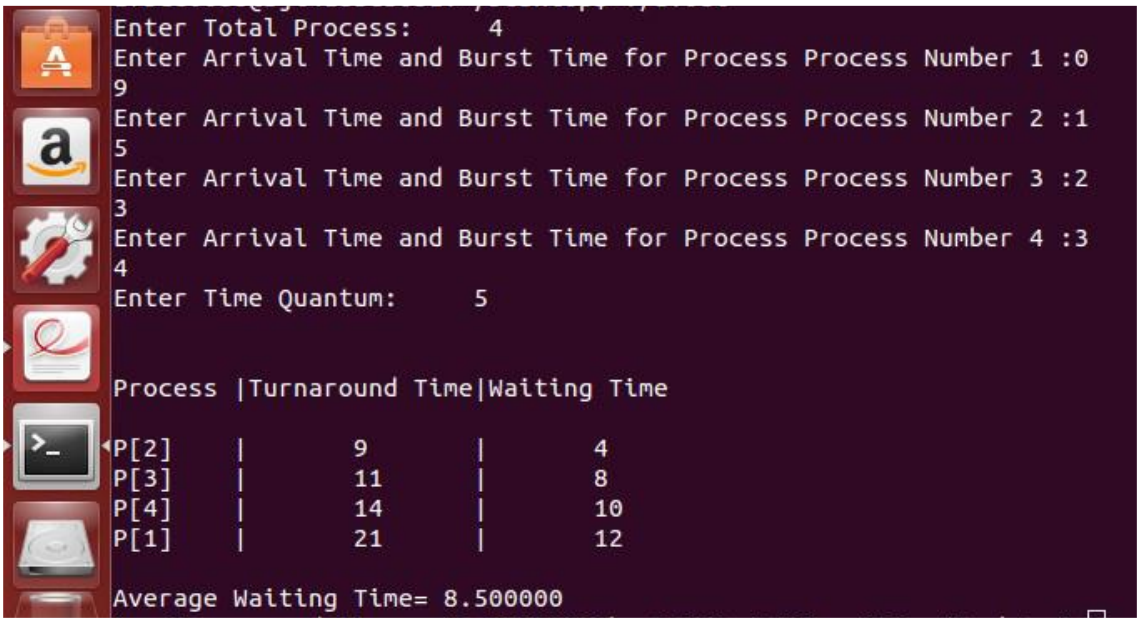
AIM

Given n processes with their burst times and priorities, the task is to find average waiting time and average turn around time using ROUND ROBIN scheduling algorithm.

CODE FOR PERFORMING ROUND ROBIN

```
#include<stdio.h>
int main()
{
	int count,j,n,time,remain,flag=0,time_quantum;
	int wait_time=0,turnaround_time=0,at[10],bt[10],rt[10];
	printf("Enter Total Process:\t ");
	scanf("%d",&n);
	remain=n;
	for(count=0;count<n;count++)
	{
		printf("Enter Arrival Time and Burst Time for Process Process
		Number %d :",count+1);
		scanf("%d",&at[count]);
		scanf("%d",&bt[count]);
		rt[count]=bt[count];
	}
	printf("Enter Time Quantum:\t");
	scanf("%d",&time_quantum);
	printf("\n\nProcess\t|Turnaround Time|Waiting Time\n\n");
	for(time=0,count=0;remain!=0;)
	{
		if(rt[count]<=time_quantum && rt[count]>0)
		{
		time+=rt[count];
		rt[count]=0;
```

```
            flag=1;
            }
            else if(rt[count]>0)
            {
            rt[count]-=time_quantum;
            time+=time_quantum;
            }
            if(rt[count]==0 && flag==1)
            {
            remain--;
            printf("P[%d]\t|\t%d\t|\t%d\n",count+1,time-at[count],time-
            at[count]-bt[count]);
            wait_time+=time-at[count]-bt[count];
            turnaround_time+=time-at[count];
            flag=0;
            }
            if(count==n-1)
            count=0;
            else if(at[count+1]<=time)
            count++;
            else
            count=0;
      }
      printf("\nAverage Waiting Time= %f\n",wait_time*1.0/n);
      printf("Avg Turnaround Time = %f",turnaround_time*1.0/n);
      return 0;
}
```



FORK SYSTEM CALLS

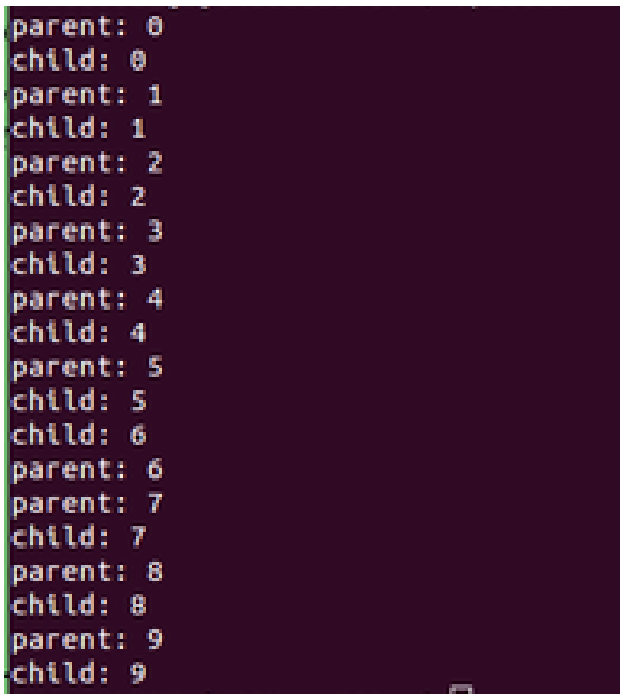
AIM

To write a program to create a child process using the system calls –fork.

CODE FOR PROCESS CREATION USING FORK

```
#include <stdio.h>
#include<sys/types.h>
int main(void)
{

int pid;

pid = fork();

if(pid == 0)
{

int j;
for(j=0; j < 10; j++)
{
printf("child: %d\n", j);
sleep(1);
}
_exit(0);
}
else if(pid > 0)
{
int i;
for(i=0; i < 10; i++)
{
printf("parent: %d\n", i);
sleep(1);
}
}
else
{
fprintf(stderr, "couldn't fork");
exit(1);
}
}
~
~
~
```

OUTPUT



AIM

To write a program experiencing the Orphan Process scenario.

Orphan Process

A process whose parent process no more exists i.e. either finished or terminated without waiting for its child process to terminate is called an orphan process. In the following code, parent finishes execution and exits while the child process is still executing and is called an orphan process now.

CODE FOR ORPHAN PROCESS

```
#include <stdio.h>
main()
{
    int pid;
    printf("I'm the original process with PID %d and PPID %d.\n",
           getpid(),getppid());
    pid=fork();  /* Duplicate. Child and parent continue from here.*/
    if (pid!=0)  /* Branch based on return value from fork() */
        {  /* pid is non-zero, so I must be the parent  */
           printf("I'm the parent process with PID %d and PPID %d.\n",
                  getpid(),getppid());
           printf("My child's PID is %d.\n", pid);
        }
    else
        {  /* pid is zero, so I must be the child. */
           sleep(5); /*Make sure that the parent terminates first. */
           printf("I'm the child process with PID %d and PPID %d.\n",
                  getpid(),getppid());
        }
    printf("PID %d terminates.\n",pid);  /* Both processes execute this */
}

~
"satyammmm31.c" 22L, 836C                                 1,1           All
```

OUTPUT

```
I'm the original process with PID 14793 and PPID 14255.
I'm the parent process with PID 14793 and PPID 14255.
My child's PID is 14794.
PID 14794 terminates.
17bce0581@sjt418scs046:~$ I'm the child process with PID 14794 and PPID 7589.
PID 0 terminates.
```

AIM

To write a program experiencing the Zombie Process scenario.

Zombie Process

A process which has finished the execution but still has entry in the process table to report to its parent process is known as a zombie process. A child process always first becomes a zombie before being removed from the process table. The parent process reads the exit status of the child process which reaps off the child process entry from the process table.

CODE FOR ZOMBIE PROCESS

```
#include <stdio.h>
main()
{
    int pid, status, childPid;
    printf("I'm the parent process and my PID is %d\n", getpid());
    pid=fork();  /* Duplicate.*/
    if (pid!=0)  /* Branch based on return value from fork() */
        {
           printf("I'm the parent process with PID %d and PPID %d.\n",
                   getpid(),getppid());
           childPid=wait(&status); /* wait for a child to terminate */
           printf("A child with PID %d terminated with exit code %d\n",
                  childPid, status>>8);
        }
    else
        {
           printf("I'm the child process with PID %d and PPID %d.\n",
                   getpid(),getppid());
          /* exit(0);  Exit with a silly number. */
        }
    printf("PID %d terminates.\n",pid);
}
~
"satyammmmm31.c" 22L, 773C                                    1,1           All
```

OUTPUT

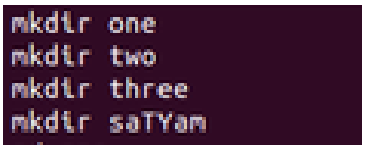
```
I'm the parent process and my PID is 14960
I'm the parent process with PID 14960 and PPID 14255.
I'm the child process with PID 14961 and PPID 14960.
PID 0 terminates.
A child with PID 14961 terminated with exit code 18
PID 14961 terminates.
```

BASIC LINUX COMMANDS

mkdir - make a directory

create new directory

```
17bce0581@sjt419scs019:~$ mkdir one
17bce0581@sjt419scs019:~$ mkdir two
17bce0581@sjt419scs019:~$ mkdir three
17bce0581@sjt419scs019:~$ mkdir saTYam
```



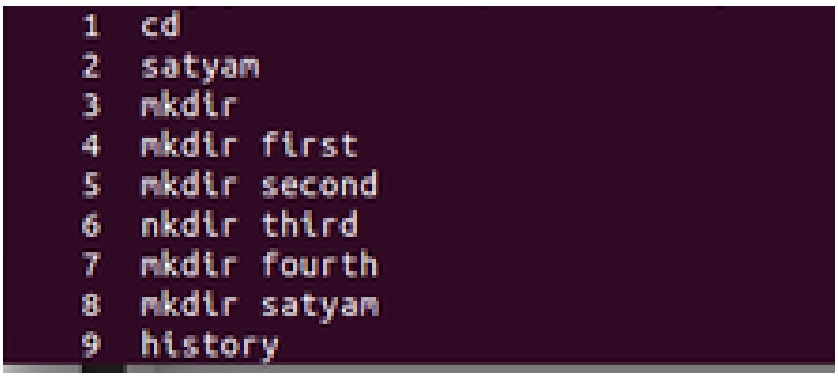
cd - change directory

Change to new directory

```
17bce0581@sjt419scs019:~$ cd one
```

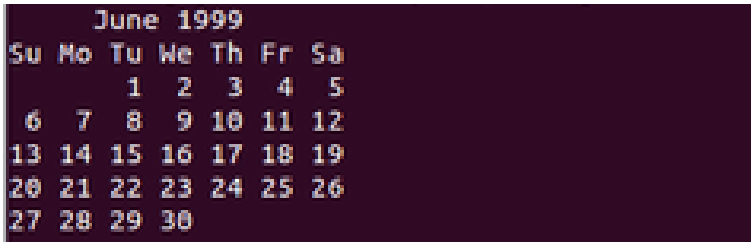
history

```
17bce0581@sjt419scs019:~/one$ history
    1  cd
    2  satyam
    3  mkdir
    4  mkdir first
    5  mkdir second
    6  nkdir third
.....
.....
   70  mv four one
   71  date
   72  history
```



cal month year

```
17bce0581@sjt419scs019:~/one$ cal june 1999
     June 1999
Su Mo Tu We Th Fr Sa
       1  2  3  4  5
 6  7  8  9 10 11 12
13 14 15 16 17 18 19
20 21 22 23 24 25 26
27 28 29 30
```



uptime

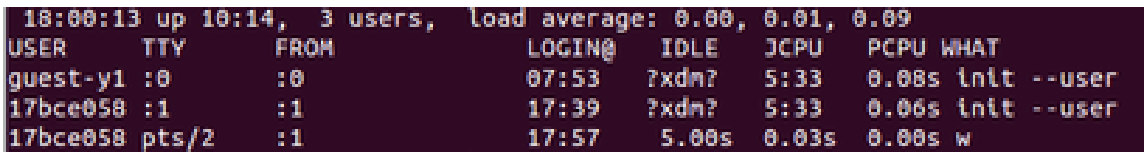
show one line summary of system status

17bce0581@sjt419scs019:~/one$ uptime
18:00:25 up 10:15, 3 users, load average: 0.00, 0.01, 0.09

w, who

who is on the system and what they are doing

17bce0581@sjt419scs019:~/one$ w
18:00:13 up 10:14, 3 users, load average: 0.00, 0.01, 0.09
USER TTY FROM LOGIN@ IDLE JCPU PCPU WHAT
guest-y1 :0 :0 07:53 ?xdm? 5:33 0.08s init --user
17bce058 :1 :1 17:39 ?xdm? 5:33 0.06s init --user
17bce058 pts/2 :1 17:57 5.00s 0.03s 0.00s w



man - display an on-line manual page