Ex No: 2

Programs using the following system calls of UNIX operating system fork, exec, getpid, exit, wait, close, stat, opendir, readdir

AIM:

To write C Programs using the following system calls of UNIX operating system fork, exec, getpid, exit, wait, close, stat, opendir, readdir.

1. PROGRAM FOR SYSTEM CALLS OF UNIX OPERATING SYSTEMS (opendir, readdir, closedir)

ALGORITHM:

```
STEP 1: Start the program.
STEP 2: Create struct dirent.
STEP 3: declare the variable buff and pointer dptr.
STEP 4: Get the directory name.
STEP 5: Open the directory.
STEP 6: Read the contents in directory and print it.
STEP 7: Close the directory.
```

PROGRAM:

```
#include<stdio.h>
#include<dirent.h>
struct dirent *dptr;
int main(int argc, char *argv[])
{
char buff[100];
DIR *dirp;
printf(“\n\n ENTER DIRECTORY NAME”);
scanf(“%s”, buff);
if((dirp=opendir(buff))==NULL)
{
printf(“The given directory does not exist”);
exit(1);
}
while(dptr=readdir(dirp))
{
printf(“%s\n”,dptr->d_name);
}
closedir(dirp);
}
```

OUTPUT:

2. PROGRAM FOR SYSTEM CALLS OF UNIX OPERATING SYSTEM (fork, getpid, exit)

ALGORITHM:

STEP 1: Start the program.
STEP 2: Declare the variables pid,pid1,pid2 .
STEP 3: Call fork() system call to create process.
STEP 4: If pid==-1, exit.
STEP 5: Ifpid!=-1 , get the process id using getpid().
STEP 6: Print the process id.
STEP 7: Stop the program

PROGRAM:

```
#include<stdio.h>
#include<unistd.h>
main()
        {
                int pid,pid1,pid2;
                pid=fork();
                if(pid==-1)
                {
                        printf(“ERROR IN PROCESS CREATION \n”);
                        exit(1);
                }
                if(pid!=0)
                {
                        pid1=getpid();
                        printf(“\n the parent process ID is %d\n”, pid1);
                }
                else
                {
                        pid2=getpid();
                        printf(“\n the child process ID is %d\n”, pid2);
                }
        }
```

OUTPUT:

RESULT:

Ex No :3 CPU SCHEDULING ALGORITHMS

1. FIRST COME FIRST SERVED (FCFS) SCHEDULING

AIM:

To write a C program for implementation of FCFS and 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.

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);
}
```

OUTPUT:

2. SHORTEST JOB FIRST (SJF) SCHEDULING

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.

PROGRAM:

```
#include<stdio.h>
#include<stdlib.h>
typedef struct
{
        int pid;
        int btime;
        int wtime;
}
        sp;
int main()
{
        int i,j,n,tbm=0,towtwtime=0,totttime
        sp*p,t;
        printf("\n sjf schaduling ..\n");
        printf("enter the no of processor");
        scanf("%d",&n);
        p=(sp*)malloc(sizeof(sp));
        printf("\n enter the burst time");
        for(i=0;i<n;i++)
        {
                printf("\n process %d\t",i+1);
                scanf("%d",&p[i].btime);
                p[i].pid=i+1;
                p[i].wtime=0;
        }
        for(i=0;i<n;i++)
        for(j=j+1,j<n;j++)
        {
                if(p[i].btime>p[j].btime)
                {
                        t=p[i];
                        p[i]=p[j];
                        p[j]=t;
                }}
        printf("\n process scheduling\n");
        printf("\n process \tburst time \t w
        for(i=0;i<n;i++)
        {
                towtwtime+=p[i].wtime=tbm;
                tbm+=p[i].btime;
                printf("\n%d\t\t%d",p[i].pid,p[i].bt;
                printf("\t\t%d\t\t%d",p[i].wtime,p[i
        }
        totttime=tbm+towtwtime;
        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);
    }
```

OUTPUT:

3. PRIORITY SCHEDULING

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.

PROGRAM:

```
#include<stdio.h>
#include<stdio.h>
#include<stdlib.h>
typedef struct
{
        int pno;
        int pri;
        int pri;
        int btime;
        int wtime;
}sp;
int main()
{
        int i,j,n;
        int tbm=0,totwtime=0,totttime=0;
        sp *p,t;
        printf("\n PRIORITY SCHEDULING.\n");
        printf("\n enter the no of process....\n");
        scanf("%d",&n);
        p=(sp*)malloc(sizeof(sp));
        printf("enter the burst time and priority:\n");
        for(i=0;i<n;i++)
        {
                printf("process%d:”,i+1);
                scanf("%d%d",&p[i].btime,&p[i].pri);
                p[i].pno=i+1;
                p[i].wtime=0;
        }
        for(i=0;i<n-1;i++)
```

```
                for(j=i+1;j<n;j++)
                {
                if(p[i].pri>p[j].pri)
                {
                        t=p[i];
                        p[i]=p[j];
                        p[j]=t;
                }
        }
        printf("\n process\tbursttime\twaiting time\tturnaround time\n");
        for(i=0;i<n;i++)
        {
                totwtime+=p[i].wtime=tbm;
                tbm+=p[i].btime;
                printf("\n%d\t\t%d",p[i].pno,p[i].btime);
                printf("\t\t%d\t\t%d",p[i].wtime,p[i].wtime+p[i].btime);
        }
        totttime=tbm+totwtime;
        printf("\n total waiting time:%d",totwtime);
        printf("\n average waiting time:%f",(float)totwtime/n);
        printf("\n total turnaround time:%d",totttime);
        printf("\n avg turnaround time:%f",(float)totttime/n);
}
```

OUTPUT:

4. ROUND ROBIN SCHEDULING

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.

PROGRAM:

```
#include<stdio.h>
#include<stdlib.h>
struct rr
{
        int pno,btime,sbtime,wtime,lst;
}p[10];
int main()
{
        int pp=-1,ts,flag,count,ptm=0,i,n,twt=0,totttime=0;
        printf("\n round robin scheduling............");
        printf("enter no of processes:");
        scanf("%d",&n);
        printf("enter the time slice:");
        scanf("%d",&ts);
        printf("enter the burst time");
        for(i=0;i<n;i++)
        {
                printf("\n process%d\t",i+1);
                scanf("%d",&p[i].btime);
                p[i].wtime=p[i].lst=0;
                p[i].pno=i+1;
                p[i].sbtime=p[i].btime;
        }
        printf("scheduling....\n");
        do
        {
                flag=0;
                for(i=0;i<n;i++)
                {
                        count=p[i].btime;
                        if(count>0)
                        {
                                flag=-1;
```

```
                                        count=(count>=ts)?ts:count;
                                        printf("\n process %d",p[i].pno);
                                        printf("from%d",ptm);
                                        ptm+=count;
                                        printf("to%d",ptm);
                                        p[i].btime-=count;
                                        if(pp!=i)
                                        {
                                                pp=i;
                                                p[i].wtime+=ptm-p[i].lst-count;
                                                p[i].lst=ptm;
                                        }
                }
```

OUTPUT:

RESULT: