CMR TECHNICAL CAMPUS

UGC AUTONOMOUS

Kandlakoya(V), Medchal Road, Hyderabad – 501 401

Accredited by NBA and NAAC with A Grade

Approved by AICTE, New Delhi and Affiliated to JNTU, Hyderabad

DEPARTMENT OF CSE (AI & ML)

OPERATING SYSTEM LAB MANUAL (R20)

COURSE CODE: 20CS406PC

Prepared by:

Mr. M. Ravindran

Assistant Professor

1. Write C programs to simulate the following CPU Scheduling algorithms
a) FCFS b) SJF c) Round Robin d) priority .

FCFS

```
#include<stdio.h>

int main()
{
int bt[20], wt[20], tat[20], i, n;
float wtavg, tatavg;
clrscr();
printf("\nEnter the number of processes -- ");
scanf("%d", &n);
for(i=0;i<n;i++)
{
printf("\nEnter Burst Time for Process %d -- ", i);
scanf("%d", &bt[i]);
}
wt[0] = wtavg = 0;
tat[0] = tatavg = bt[0];
for(i=1;i<n;i++)
{
wt[i] = wt[i-1] +bt[i-1];
tat[i] = tat[i-1] +bt[i];
wtavg = wtavg + wt[i];
tatavg = tatavg + tat[i];
}
printf("\t PROCESS \tBURST TIME \t WAITING TIME\t TURNAROUND TIME\n");
for(i=0;i<n;i++)
printf("\n\t P%d \t\t %d \t\t %d \t\t %d", i, bt[i], wt[i], tat[i]);
printf("\nAverage Waiting Time -- %f", wtavg/n);
printf("\nAverage Turnaround Time -- %f", tatavg/n);
getch();
}
```

INPUT

```
Enter the number of processes -- 3
Enter Burst Time for Process 0 -- 24
Enter Burst Time for Process 1 -- 3
Enter Burst Time for Process 2 – 3
```

OUTPUT

PROCESS	BURST TIME	WAITING TIME	TURNAROUND TIME
P0	24	0	24

P1 3 24 27
P2 3 27 30
Average Waiting Time-- 17.000000
Average Turnaround Time -- 27.000000

SJF

```
#include<stdio.h>
int main()
{
int p[20], bt[20], wt[20], tat[20], i, k, n, temp; float wtavg, tatavg;
clrscr();
printf("\nEnter the number of processes -- ");
scanf("%d", &n);
for(i=0;i<n;i++)
{
p[i]=i;
printf("Enter Burst Time for Process %d -- ", i);
scanf("%d", &bt[i]);
}
for(i=0;i<n;i++)
for(k=i+1;k<n;k++)
if(bt[i]>bt[k])
{
temp=bt[i];
bt[i]=bt[k];
bt[k]=temp;
temp=p[i];
p[i]=p[k];
p[k]=temp;
}
wt[0] = wtavg = 0;
tat[0] = tatavg = bt[0];
for(i=1;i<n;i++)
{
wt[i] = wt[i-1] +bt[i-1];
tat[i] = tat[i-1] +bt[i];
wtavg = wtavg + wt[i];
tatavg = tatavg + tat[i];
}
printf("\n\t PROCESS \tBURST TIME \t WAITING TIME\t TURNAROUND TIME\n");
for(i=0;i<n;i++)
printf("\n\t P%d \t\t %d \t\t %d \t\t %d", p[i], bt[i], wt[i], tat[i]);
printf("\nAverage Waiting Time -- %f", wtavg/n);
printf("\nAverage Turnaround Time -- %f", tatavg/n);
getch();
```

}

INPUT

Enter the number of processes -- 4
Enter Burst Time for Process 0 -- 6
Enter Burst Time for Process 1 -- 8
Enter Burst Time for Process 2 -- 7
Enter Burst Time for Process 3 – 3

OUTPUT

PROCESS	BURST TIME	WAITING TIME	TURNAROUND TIME
P3	3	0	3
P0	6	3	9
P2	7	9	16
P1	8	16	24

Average Waiting Time -- 7.000000
Average Turnaround Time -- 13.000000

Round Robin

```
#include<stdio.h>
int main()
{
int i,j,n,bu[10],wa[10],tat[10],t,ct[10],max;
float awt=0,att=0,temp=0;
clrscr();
printf("Enter the no of processes -- ");
scanf("%d",&n);
for(i=0;i<n;i++)
{
printf("\nEnter Burst Time for process %d -- ", i+1);
scanf("%d",&bu[i]);
ct[i]=bu[i];
}
printf("\nEnter the size of time slice -- ");
scanf("%d",&t);
max=bu[0];
for(i=1;i<n;i++)
if(max<bu[i])
max=bu[i];
for(j=0;j<(max/t)+1;j++)
for(i=0;i<n;i++)
if(bu[i]!=0)
if(bu[i]<=t)
{
tat[i]=temp+bu[i];
temp=temp+bu[i];
```

```
bu[i]=0;
}
else
{
bu[i]=bu[i]-t;
temp=temp+t;
}
for(i=0;i<n;i++){
wa[i]=tat[i]- ct[i];
att+=tat[i];
awt+=wa[i];
}
printf("\nThe Average Turnaround time is -- %f",att/n);
printf("\nThe Average Waiting time is -- %f ",awt/n);
printf("\n\tPROCESS\t BURST TIME \t WAITING TIME\tTURNAROUND TIME\n");
for(i=0;i<n;i++)
printf("\t%d \t %d \t\t %d \t\t %d \n",i+1,ct[i],wa[i],tat[i]);
getch();
}
```

INPUT:

Enter the no of processes – 3
Enter Burst Time for process 1 – 24
Enter Burst Time for process 2 -- 3
Enter Burst Time for process 3 – 3
Enter the size of time slice – 3

OUTPUT:

PROCESS	BURST TIME	WAITING TIME	TURNAROUNDTIME
1	24	6	30
2	3	4	7
3	3	7	10

The Average Turnaround time is – 15.666667
The Average Waiting time is -- 5.666667

Priority

```
#include<stdio.h>
int main()
{
int p[20],bt[20],pri[20], wt[20],tat[20],i, k, n, temp;
float wtavg, tatavg;
clrscr();
printf("Enter the number of processes --- ");
scanf("%d",&n);
```

```
for(i=0;i<n;i++){
p[i] = i;
printf("Enter the Burst Time & Priority of Process %d --- ",i);
scanf("%d %d",&bt[i], &pri[i]);
}
for(i=0;i<n;i++)
for(k=i+1;k<n;k++)
if(pri[i] > pri[k])
{
temp=p[i];
p[i]=p[k];
p[k]=temp;
temp=bt[i];
bt[i]=bt[k];
bt[k]=temp;
temp=pri[i];
pri[i]=pri[k];
pri[k]=temp;
}
wtavg = wt[0] = 0;
tatavg = tat[0] = bt[0];
for(i=1;i<n;i++)
{
wt[i] = wt[i-1] + bt[i-1];
tat[i] = tat[i-1] + bt[i];
wtavg = wtavg + wt[i];
tatavg = tatavg + tat[i];
}
printf("\nPROCESS\t\tPRIORITY\tBURST TIME\tWAITING TIME\tTURNAROUND
TIME");
for(i=0;i<n;i++)
printf("\n%d \t\t %d \t\t %d \t\t %d \t\t %d ",p[i],pri[i],bt[i],wt[i],tat[i]);
printf("\nAverage Waiting Time is --- %f",wtavg/n);
printf("\nAverage Turnaround Time is --- %f",tatavg/n);
getch();
}
```

INPUT

```
Enter the number of processes -- 5
Enter the Burst Time & Priority of Process 0 --- 10     3
Enter the Burst Time & Priority of Process 1 --- 1      1
Enter the Burst Time & Priority of Process 2 --- 2      4
Enter the Burst Time & Priority of Process 3 --- 1      5
Enter the Burst Time & Priority of Process 4 --- 5      2
```

OUTPUT

PROCESS	PRIORITY	BURST TIME	WAITING TIME	TURNAROUND TIME
1	1	1	0	1
4	2	5	1	6
0	3	10	6	16
2	4	2	16	18
3	5	1	18	19

Average Waiting Time is --- 8.200000
Average Turnaround Time is --- 12.000000

2. Write programs using the I/O system calls of UNIX/LINUX operating system (open, read, write, close, fcntl, seek, stat, opendir, readdir)

open, read, write, close

```
#include <sys/types.h>
#include<stdio.h>
#include <unistd.h>
#include <fcntl.h>
#define BUFSIZE 512
int main ()
{
int from, to, nr, nw, n;
char buf[BUFSIZE],ch;
if ((from=open("one.txt", O_RDONLY)) < 0) {
          printf("Error opening source file");
          exit(1);
     }

     if ((to=creat("two.txt", O_RDWR)) < 0) {
          printf("Error creating destination file");
          exit(2);
     }
while((nr=read(from, buf, sizeof( buf))) != 0) {
       if (nr < 0) {
          printf("Error reading source file");
    exit(3);

       }
       nw=0;
       do {
         if ((n=write(to, &buf[nw], nr-nw)) < 0) {
           printf("Error writing destination file");
           exit(4);
         }
         nw += n;
       } while (nw < nr);
     }
printf("successfully copied the content from fiel one.txt to two.txt");

     close(from); close(to);
}
```

INPUT
Create file name one.txt with some content

OUTPUT
Successfully copied the content from file one.txt to two.txt
//you can see that program has created file two.txt and has content same as one.txt.

opendir, readdir

```
#include<stdio.h>
#include<dirent.h>
int main()
{
	struct dirent *de;
	DIR *dr=opendir(".");
	if(dr==NULL)
	{
		printf("Could not open current Directory");
		return 0;
	}
	while((de=readdir(dr))!=NULL)
			printf("%s\n",de->d_name);
	closedir(dr);
	return 1;
	}
```

OUTPUT
// will be list of all the directories; example output below:
ashrc
.bash_history
.bash_logout
.fcfs.c.swp
.landscape
.motd_shown
.msgqs.c.swp
.msgsend.c.swp
.pro.c.swp
.profile
.sc.c.swp
.sudo_as_admin_successful
.viminfo
a.c
a.out
dir
dir.c
file1
file1.c
file2
file2.c

hello.c
mqr.c
mqw.c
msgq.txt
msgrecv.c
stat

```
#include<stdio.h>
#include<sys/stat.h>
int main()
{
	struct stat sfile;
	stat("stat.c",&sfile);
	printf("st_mode=%o", sfile.st_mode);
	return 0;
}
```

OUTPUT
st_mode=100644

lseek

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <fcntl.h>

void func(char arr[], int n)
{
	int f_write = open("start.txt", O_RDONLY);

	int f_read = open("end.txt", O_WRONLY);

	int count = 0;
	while (read(f_write, arr, 1))
	{
		if (count < n)
		{
			lseek (f_write, n, SEEK_CUR);
			write (f_read, arr, 1);
			count = n;
		}

		else
		{
			count = (2*n);
```

```
                        lseek(f_write, count, SEEK_CUR);
                        write(f_read, arr, 1);
                }
        }
        close(f_write);
        close(f_read);
}

int main()
{
        char arr[100];
        int n;
        n = 5;
        func(arr, n);
        return 0;
}
```

OUTPUT
//Create two file start.txt and end.txt write the content in one.txt.
You can see the sleek output in end.txt.

fcntl-creating a write lock

```
#include <fcntl.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>

int main (int argc, char* argv[])
{
 char* file = argv[1];
 int fd;
 struct flock lock;

 printf ("opening %s\n", file);

 fd = open (file, O_WRONLY);
 printf ("locking\n");

 memset (&lock, 0, sizeof(lock));
 lock.l_type = F_WRLCK;

 fcntl (fd, F_SETLKW, &lock);

 printf ("locked; hit Enter to unlock... ");

 getchar ();
```

```
 printf ("unlocking\n");

 lock.l_type = F_UNLCK;
 fcntl (fd, F_SETLKW, &lock);

 close (fd);
 return 0;
}
```

OUTPUT

```
opening NULL
locking
locked; hit Enter to unlock...
```

3. Write a C program to simulate Bankers Algorithm for Deadlock Avoidance and Prevention.

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
int main()
{
int alloc[10][10],max[10][10];
int avail[10],work[10],total[10];
int i,j,k,n,need[10][10];
int m;
int count=0,c=0;
char finish[10];
clrscr();
printf("Enter the no. of processes and resources:");
scanf("%d%d",&n,&m);
for(i=0;i<=n;i++)
finish[i]='n';
printf("Enter the claim matrix:\n");
for(i=0;i<n;i++)
for(j=0;j<m;j++)
scanf("%d",&max[i][j]);
printf("Enter the allocation matrix:\n");
for(i=0;i<n;i++)
for(j=0;j<m;j++)
scanf("%d",&alloc[i][j]);
printf("Resource vector:");
for(i=0;i<m;i++)
scanf("%d",&total[i]);
for(i=0;i<m;i++)
avail[i]=0; for(i=0;i<n;i++)
for(j=0;j<m;j++)
avail[j]+=alloc[i][j];
for(i=0;i<m;i++)
work[i]=avail[i];
for(j=0;j<m;j++)
work[j]=total[j]-work[j];
for(i=0;i<n;i++)
for(j=0;j<m;j++)
need[i][j]=max[i][j]-alloc[i][j];
A:
for(i=0;i<n;i++)
{
c=0;
for(j=0;j<m;j++)
if((need[i][j]<=work[j])&&(finish[i]=='n'))
c++;
```

```
if(c==m)
{
printf("All the resources can be allocated to Process %d", i+1);
printf("\n\nAvailable resources are:");
for(k=0;k<m;k++)
{
work[k]+=alloc[i][k];
printf("%4d",work[k]);
}
printf("\n");
finish[i]='y';
printf("\nProcess %d executed?:%c \n",i+1,finish[i]);
count++;
}
}
if(count!=n)
goto A;
else
printf("\n System is in safe mode");
printf("\n The given state is safe state");
getch();
}
```

OUTPUT

```
Enter the no. of processes and resources: 4 3
Enter the claim matrix:
3 2 2
6 1 3
3 1 4
4 2 2
Enter the allocation matrix:
1 0 0
6 1 2
2 1 1
0 0 2
Resource vector:9 3 6
All the resources can be allocated to Process 2
Available resources are: 6 2 3
Process 2 executed?:y
All the resources can be allocated to Process 3 Available resources
are: 8 3 4
Process 3 executed?:y
All the resources can be allocated to Process 4 Available resources
are: 8 3 6
Process 4 executed?:y
A
ll the resources can be allocated to Process 1
```

Available resources are: 9 3 6
Process 1 executed?:y
System is in safe mode
The given state is safe state

4. Write a C program to implement the Producer – Consumer problem using semaphores using UNIX/LINUX system calls.

```
#include<stdio.h>
int main()
{
int buffer[10], bufsize, in, out, produce, consume, choice=0; in = 0;
out = 0;
bufsize = 10;
while(choice !=3)
{
printf("\n1. Produce \t 2. Consume \t3. Exit");
printf("\nEnter your choice: ");
scanf("%d",&choice);
switch(choice)
{
case 1: if((in+1)%bufsize==out)
printf("\nBuffer is Full");
else
{
}
break;;;
printf("\nEnter the value: ");
scanf("%d", &produce);
buffer[in] = produce;
in = (in+1)%bufsize;
case 2: if(in == out)
printf("\nBuffer is Empty");
else
{
consume = buffer[out];
printf("\nThe consumed value is %d", consume);
out = (out+1)%bufsize;
}
break;
}
}
}
```

OUTPUT

```
1. Produce
2. Consume
3. Exit
Enter your choice:
2
Buffer is Empty
```

1. Produce
2. Consume
3. Exit
Enter your choice:
1
Enter the value: 100
1. Produce
2. Consume
3. Exit
Enter your choice: 2
The consumed value is 100
1. Produce
2. Consume
3. Exit
Enter your choice: 3

5.Write C programs to illustrate the following IPC mechanisms
a) Pipes b) FIFOs c) Message Queues d) Shared Memory

Pipes

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<sys/types.h>
#include<sys/wait.h>
#include<unistd.h>
int main()
{
	int fd1[2];
	int fd2[2];
	char fixed_str[]="Welcome";
	char input_str[100];
	pid_t p;
	if(pipe(fd1)==-1){
		fprintf(stderr,"pipe failed");
		return 1;
	}
	if(pipe(fd2)==-1){
		fprintf(stderr,"pipe failed");
		return 1;
	}
	scanf("%s",input_str);
	p=fork();
	if(p<0){
		fprintf(stderr, "fork failed");
		return 1;
	}
	else if (p>0){
		char concat_str[100];
		close(fd1[0]);
		write(fd1[1],input_str,strlen(input_str)+1);
		close(fd1[1]);
		wait(NULL);
		close(fd2[1]);
		read(fd2[0],concat_str,  100);
		printf("concatenated string %s\n",concat_str);
		close(fd2[0]);
	}
	else{
		close(fd1[1]);
		char concat_str[100];
		read(fd1[0],concat_str,100);
		int k=strlen(concat_str);
```

```
        int i;
        for(i=0;i<strlen(fixed_str);i++)
            concat_str[k++]=fixed_str[i];
        concat_str[k]='\0';
        close(fd1[0]);
        close(fd2[0]);
        write(fd2[1], concat_str, strlen(concat_str)+1);
        close(fd2[1]);
        exit(0);
    }
}
```

OUTPUT

```
ubuntu@DESKTOP-B8CV2UR:~$ ./a.out
Hi
concatenated string HiWelcome
ubuntu@DESKTOP-B8CV2UR:~$
```

FIFOs

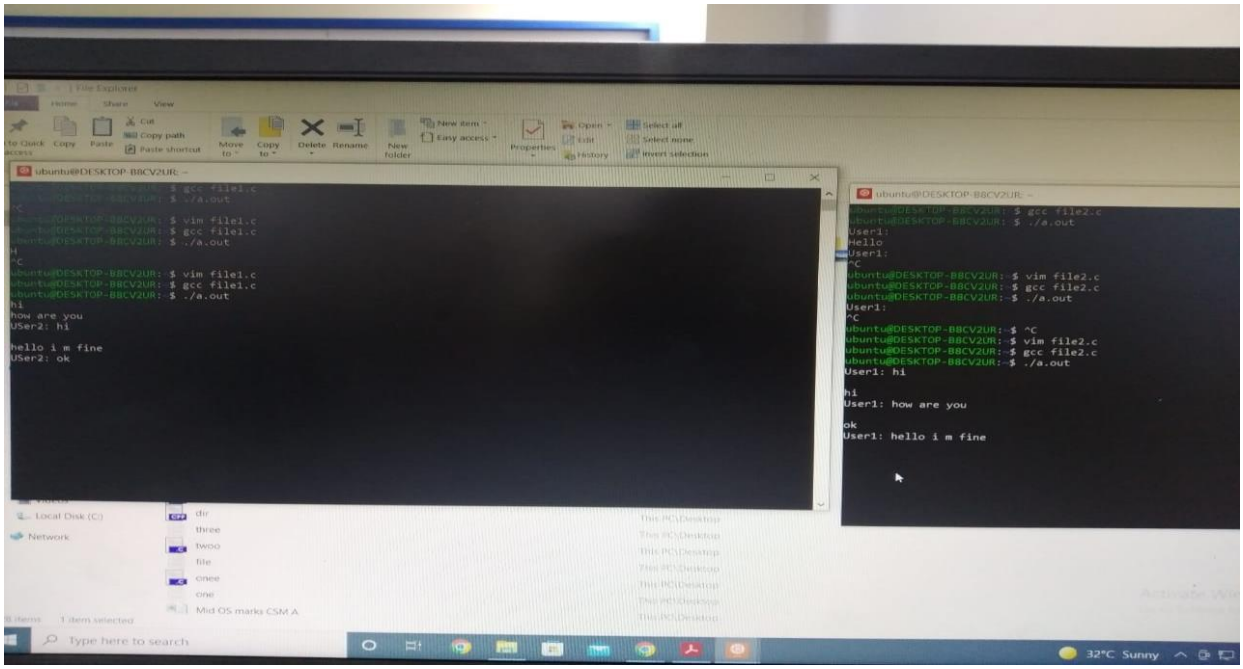
Fifo writer code

```
#include<stdio.h>
#include<string.h>
#include<fcntl.h>
#include<sys/stat.h>
#include<sys/types.h>
#include<unistd.h>
int main()
{
    int fd;
    char *myfifo="/home/ubuntu/myfifo";
    mkfifo(myfifo,0666);
    char arr1[80], arr2[80];
    while(1)
    {
        fd=open(myfifo,O_WRONLY);
        fgets(arr2, 80, stdin);
        write(fd, arr2,strlen(arr2)+1);
        close(fd);
        fd=open(myfifo,O_RDONLY);
        read(fd,arr1,sizeof(arr1));
        printf("USer2: %s\n",arr1);
        close(fd);
    }
    return 0;
}
```

Fifo readers code

```
#include<stdio.h>
#include<string.h>
#include<fcntl.h>
#include<sys/stat.h>
#include<sys/types.h>
#include<unistd.h>
int main()
{
	int fd;
	char * myfifo="/home/ubuntu/myfifo";
	mkfifo(myfifo,0666);
	char str1[80],str2[80];
	while(1)
	{
		fd=open(myfifo, O_RDONLY);
		read(fd,str1,80);
		printf("User1: %s\n",str1);
		close(fd);
		fd=open(myfifo, O_WRONLY);
		fgets(str2,80,stdin);
		write(fd,str2,strlen(str2)+1);
		close(fd);
	}
	return 0;
}
```

OUTPUT
//simultaneously execute both write and reader code in two terminals



Message Queues

//sender code

```
#include<stdlib.h>
 #include<stdio.h>
 #include<string.h>
 #include<unistd.h>
 #include<sys/types.h>
 #include<sys/ipc.h>
 #include<sys/msg.h>
 #define MAX_TEXT 512
 struct my_msg{
        long int msg_type;
        char some_text[MAX_TEXT];
 };
 int main()
 {
        int running=1;
        int msgid;
        struct my_msg some_data;
        char buffer[50];
        msgid=msgget((key_t)14534,0666|IPC_CREAT);
        if (msgid == -1)
        {
                printf("Error in creating queue\n");
                exit(0);
```

```
        }

        while(running)
        {
                printf("Enter some text:\n");
                fgets(buffer,50,stdin);
                some_data.msg_type=1;
                strcpy(some_data.some_text,buffer);
                if(msgsnd(msgid,(void *)&some_data, MAX_TEXT,0)==-1)
                {
                        printf("Msg not sent\n");
                }
                if(strncmp(buffer,"end",3)==0)
                {
                        running=0;
                }
        }
}
//Receiver code
#include<stdlib.h>
 #include<stdio.h>
 #include<string.h>
 #include<unistd.h>
 #include<sys/types.h>
 #include<sys/ipc.h>
 #include<sys/msg.h>
 struct my_msg{
        long int msg_type;
        char some_text[BUFSIZ];
 };
 int main()
 {
        int running=1;
        int msgid;
        struct my_msg some_data;
        long int msg_to_rec=0;
        msgid=msgget((key_t)12345,0666|IPC_CREAT);
        while(running)
        {
                msgrcv(msgid,(void *)&some_data,BUFSIZ,msg_to_rec,0);
                printf("Data received: %s\n",some_data.some_text);
                if(strncmp(some_data.some_text,"end",3)==0)
                {
```

```
                running=0;
            }
        }
        msgctl(msgid,IPC_RMID,0);
}
```

OUTPUT

// Execution is same as as FIFO

Shared Memory

//Writer code

```
#include<stdlib.h>
#include<string.h>
#include<sys/shm.h>
#include<stdio.h>
#include<unistd.h>
int main()
{
    int i;
    void * shared_memory;
    char buff[100];
    int shmid;
    shmid=shmget((key_t)2345,1024,0666|IPC_CREAT);
    printf("key of shared memory is %d\n",shmid);
    shared_memory=shmat(shmid,NULL,0);
    printf("Process attached at %p\n",shared_memory);
    printf("Enter some data to write to shared memory\n");
    read(0,buff,100);
    strcpy(shared_memory,buff);
    printf("you wrote:%s\n",(char*)shared_memory);
}
```

//Readers Code

```
#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("keyof sharedmemory is %d\n",shmid);
    shared_memory=shmat(shmid,NULL,0);
    printf("process attacehed at %p\n", shared_memory);
    printf("Data read form shared memroyis %s\n",(char*)shared_memory);
}
```

OUTPUT

6. Write C programs to simulate the following memory management techniques
a) Paging b) Segmentation

Paging

```
#include<stdio.h>
#include<conio.h>
int main()
{
 int ms, ps, nop, np, rempages, i, j, x, y, pa, offset;
 int s[10], fno[10][20];
printf("\nEnter the memory size -- ");
scanf("%d",&ms);
printf("\nEnter the page size -- ");
scanf("%d",&ps);
nop = ms/ps;
printf("\nThe no. of pages available in memory are -- %d ",nop);
printf("\nEnter number of processes -- ");
 scanf("%d",&np);
rempages = nop;
for(i=1;i<=np;i++)
{
printf("\nEnter no. of pages required for p[%d]-- ",i);
 scanf("%d",&s[i]);
if(s[i] >rempages)
{
printf("\nMemory is Full");
break;
}
rempages = rempages - s[i];
printf("\nEnter pagetable for p[%d] --- ",i);
 for(j=0;j<s[i];j++)
scanf("%d",&fno[i][j]);
}
printf("\nEnter Logical Address to find Physical Address ");
printf("\nEnter process no. and pagenumber and offset -- ");
scanf("%d %d %d",&x,&y, &offset);
if(x>np || y>=s[i] || offset>=ps)
printf("\nInvalid Process or Page Number or offset");
else
{
pa=fno[x][y]*ps+offset;
```

```
printf("\nThe Physical Address is -- %d",pa);
}
getch();
}
```

OUTPUT

```
Enter the memory size – 1000 Enter the page size -- 100
The no. of pages available in memory are -- 10
Enter number of processes -- 3
Enter no. of pages required for p[1]-- 4
Enter pagetable for p[1] --- 8 6
9
5
Enter no. of pages required for p[2]-- 5
Enter pagetable for p[2] --- 1 4 5 7 3
Enter no. of pages required for p[3]—5


Memory is Full
Enter Logical Address to find Physical Address Enter process no. and pagenumber and offset  --
2
3
60
The Physical Address is --  760
```

Segmentation

```
#include<stdio.h>
#include<stdlib.h>
#include<conio.h>
struct list
{
int seg;
int base;
int limit;
struct list *next;
} *p;
void insert(struct list *q,int base,int limit,int seg)
{
if(p==NULL)
{
p=(struct list*)malloc(sizeof(struct list));
p->limit=limit;
p->base=base;
p->seg=seg;
```

```
p->next=NULL;
}
else
{
while(q->next!=NULL)
{
q=q->next;
printf("yes");
}
q->next= (struct list*)malloc(sizeof(struct list));
q->next ->limit=limit;
q->next ->base=base;
q->next ->seg=seg;
q->next ->next=NULL;
}
}
int find(struct list *q,int seg)
{
while(q->seg!=seg)
{
q=q->next;
}
return q->limit;
}
int search(struct list *q,int seg)
{
while(q->seg!=seg)
{
q=q->next;
}
return q->base;
}
int main()
{
p=NULL;
int seg,offset,limit,base,c,s,physical;
printf("Enter segment table/n");
printf("Enter -1 as segment value for termination\n");
do
{
printf("Enter segment number");
scanf("%d",&seg);
if(seg!=-1)
{
printf("Enter base value:");
scanf("%d",&base);
```

```
printf("Enter value for limit:");
scanf("%d",&limit);
insert(p,base,limit,seg);
}
}
while(seg!=-1);
printf("Enter offset:");
scanf("%d",&offset);
printf("Enter bsegmentation number:");
scanf("%d",&seg);
c=find(p,seg);
s=search(p,seg);
if(offset<c)
{
physical=s+offset;
printf("Address in physical memory %d\n",physical);
}
else
{
printf("error");
}

}
```

OUTPUT

```
Enter segment table
Enter -1 as segmentation value for termination
Enter segment number:1
Enter base value:2000
Enter value for limit:100
Enter segment number:2
Enter base value:2500
Enter value for limit:100
Enter segmentation number:-1
Enter offset:90
Enter segment number:2
Address in physical memory 2590
```