

OS Lab Manual

Experiment - 1 :

Aim : Write a C program to simulate the following non-preemptive CPU scheduling algorithms to find turnaround time and waiting time for the above problem.

a) FCFS b) SJF c) Round Robin d) Priority

a)FCFS CPU SCHEDULING ALGORITHM :

PROGRAM:

```
#include<stdio.h>
main()
{
    int bt[20], wt[20], tat[20], i, n;
    float wtavg, tatavg;
    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];
    }
}
```

Reg.No:

```
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);  
}
```

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

b)SJF CPU SCHEDULING ALGORITHM :

PROGRAM:

```
#include<stdio.h>
main()
{
    int p[20], bt[20], wt[20], tat[20], i, k, n, temp;
    float wtavg, tatavg;
    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];
    }
```

Reg.No:

```
printf("\n\t PROCESS \t BURST TIME \t WAITING TIME \t TURNAROUND  
TIME\n");  
for(i=0;i<n;i++)  
    printf("\n\t P%d \t %d \t %d \t %d", p[i], bt[i], wt[i], tat[i]);  
    printf("\nAverage Waiting Time -- %f", wtavg/n);  
    printf("\nAverage Turnaround Time -- %f", tatavg/n);  
}
```

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 TIME	BURST TIME	WAITING TIME	TURNAROUND
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

c)ROUND ROBIN CPU SCHEDULING ALGORITHM

PROGRAM:

```
#include<stdio.h>
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;
```

Reg.No:

```
    }  
    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 %d \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:

The Average Turnaround time is – 15.666667

The Average Waiting time is -- 5.666667

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

d)PRIORITY CPU SCHEDULING ALGORITHM

PROGRAM :

```
#include<stdio.h>
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];
```

Reg.No:

```
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

Experiment - 2 :

Aim :

Write a C program to simulate the MVT and MFT memory management techniques

a)MFT MEMORY MANAGEMENT TECHNIQUE :

PROGRAM :

```
#include<stdio.h>
main()
{
    int ms, bs, nob, ef,n, mp[10],tif=0;
    int i,p=0;
    printf("Enter the total memory available (in Bytes) -- ");
    scanf("%d",&ms);
    printf("Enter the block size (in Bytes) -- ");
    scanf("%d", &bs);
    nob=ms/bs;
    ef=ms - nob*bs;
    printf("\nEnter the number of processes -- ");
    scanf("%d",&n);
    for(i=0;i<n;i++)
    {
        printf("Enter memory required for process %d (in Bytes)-- ",i+1);
        scanf("%d",&mp[i]);
    }
    printf("\nNo. of Blocks available in memory -- %d",nob);
    printf("\n\nPROCESS\tMEMORY REQUIRED\tALLOCATED\tINTERNAL\nFRAGMENTATION");

    for(i=0;i<n && p<nob;i++)
    {
```

Reg.No:

```
printf("\n %d\t\t%d",i+1,mp[i]);
if(mp[i] > bs)
printf("\t\tNO\t\t---");
else
{
    printf("\t\tYES\t\t%d",bs-mp[i]);
    tif = tif + bs-mp[i];
    p++;
}
}
if(i<n)
    printf("\nMemory is Full, Remaining Processes cannot be accomodated");
printf("\n\nTotal Internal Fragmentation is %d",tif);
printf("\nTotal External Fragmentation is %d",ef);
}
```

INPUT :

Enter the total memory available (in Bytes) -- 1000
Enter the block size (in Bytes)-- 300
Enter the number of processes – 5
Enter memory required for process 1 (in Bytes) -- 275
Enter memory required for process 2 (in Bytes) -- 400
Enter memory required for process 3 (in Bytes) -- 290
Enter memory required for process 4 (in Bytes) -- 293
Enter memory required for process 5 (in Bytes) -- 100
No. of Blocks available in memory -- 3

OUTPUT:

PROCESS	MEMORY REQUIRED	ALLOCATED	INTERNAL FRAGMENTATION
1	275	YES	25
2	400	NO	-----
3	290	YES	10
4	293	YES	7

Memory is Full, Remaining Processes cannot be accommodated
Total Internal Fragmentation is 42
Total External Fragmentation is 100

b)MVT MEMORY MANAGEMENT TECHNIQUE :

PROGRAM :

```
#include<stdio.h>
main()
{
    int ms,mp[10],i, temp,n=0;
    char ch = 'y';
    printf("\nEnter the total memory available (in Bytes)-- ");
    scanf("%d",&ms);
    temp=ms;
    for(i=0;ch=='y';i++,n++)
    {
        printf("\nEnter memory required for process %d (in Bytes) -- ",i+1);
        scanf("%d",&mp[i]);
        if(mp[i]<=temp)
        {
            printf("\nMemory is allocated for Process %d ",i+1);
            temp = temp - mp[i];
        }
        else
        {
            printf("\nMemory is Full");
            break;
        }
        printf("\nDo you want to continue(y/n) -- ");
        scanf(" %c", &ch);
    }
    printf("\n\nTotal Memory Available -- %d", ms);
    printf("\n\n\tPROCESS\t\tMEMORY ALLOCATED ");
    for(i=0;i<n;i++)
        printf("\n \t%d\t\t%d",i+1,mp[i]);
    printf("\n\nTotal Memory Allocated is %d",ms-temp);
    printf("\nTotal External Fragmentation is %d",temp);
}
```

Reg.No:

INPUT:

Enter the total memory available (in Bytes) -- 1000
Enter memory required for process 1 (in Bytes) -- 400
Memory is allocated for Process 1
Do you want to continue(y/n) -- y
Enter memory required for process 2 (in Bytes) -- 275
Memory is allocated for Process 2
Do you want to continue(y/n) -- y
Enter memory required for process 3 (in Bytes) -- 550

OUTPUT:

Memory is Full
Total Memory Available -- 1000

PROCESS	MEMORY ALLOCATED
1	400
2	275

Total Memory Allocated is 675
Total External Fragmentation is 325

Experiment - 3:

Aim: Write a program to implement BANKER'S ALGORITHM

PROGRAM:

```
#include<stdio.h>

struct da
{
    int max[10],a1[10],need[10],before[10],after[10];
}p[10];

void main()
{
    int i,j,k,l,r,n,tot[10],av[10],cn=0,cz=0,temp=0,c=0;

    printf("\n ENTER THE NO. OF PROCESSES:");

    scanf("%d",&n);

    printf("\n ENTER THE NO. OF RESOURCES:");

    scanf("%d",&r);

    for(i=0;i<n;i++)
    {
        printf("PROCESS %d \n",i+1);

        for(j=0;j<r;j++)
        {
            printf("MAXIMUM VALUE FOR RESOURCE %d:",j+1);

            scanf("%d",&p[i].max[j]);
        }
    }
}
```

Reg.No:

```
for(j=0;j<r;j++)
{
    printf("ALLOCATED FROM RESOURCE %d:",j+1);
    scanf("%d",&p[i].a1[j]);
    p[i].need[j]=p[i].max[j]-p[i].a1[j];
}
}

for(i=0;i<r;i++)
{
    printf("ENTER TOTAL VALUE OF RESOURCE %d:",i+1);
    scanf("%d",&tot[i]);
}

for(i=0;i<r;i++)
{
    for(j=0;j<n;j++)
    {
        temp=temp+p[j].a1[i];
        av[i]=tot[i]-temp;
        temp=0;
    }
}

printf("\n\t RESOURCES  ALLOCATED  NEEDED  TOTAL  AVAIL");
```

Reg.No:

```
for(i=0;i<n;i++)
{
    printf("\n P%d \t",i+1);

    for(j=0;j<r;j++)

        printf("%d",p[i].max[j]);

        printf("\t");

    for(j=0;j<r;j++)

        printf("%d",p[i].a1[j]);

        printf("\t");

    for(j=0;j<r;j++)

        printf("%d",p[i].need[j]);

        printf("\t");

    for(j=0;j<r;j++)
    {

        if(i==0)

            printf("%d",tot[j]);

    }

    printf("");

    for(j=0;j<r;j++)

    {

        if(i==0)

            printf("%d",av[j]);

    }

}
```

Reg.No:

```
}

printf("\n\n\t A VAIL  BEFORE\t A VAIL AFTER ");

for(l=0;l<n;l++)

{

    for(i=0;i<n;i++)

    {

        for(j=0;j<r;j++)

        {

            if(p[i].need[j] > av[j])

                cn++;

            if(p[i].max[j]==0)

                cz++;

        }

        if(cn==0 && cz!=r)

        {

            for(j=0;j<r;j++)

            {

                p[i].before[j]=av[j]-p[i].need[j];

                p[i].after[j]=p[i].before[j]+p[i].max[j];

                av[j]=p[i].after[j];

                p[i].max[j]=0;

            }

            printf("\n P %d \t",i+1);
```


Reg.No:

```
        for(j=0;j<r;j++)
            printf("%d",p[i].before[j]);

        printf("\t");

        for(j=0;j<r;j++)
            printf("%d",p[i].after[j]);

        cn=0;

        cz=0;

        c++;

        break;
    }

    else

    {

        cn=0;cz=0;

    }

}

}

if(c==n)

printf("\n THE ABOVE SEQUENCE IS A SAFE SEQUENCE");

    else

        printf("\n DEADLOCK OCCURED");

        getch();

}
```

INPUT:

//TEST CASE 1:

ENTER THE NO. OF PROCESSES:4

ENTER THE NO. OF RESOURCES:3

PROCESS 1

MAXIMUM VALUE FOR RESOURCE 1:3

MAXIMUM VALUE FOR RESOURCE 2:2

MAXIMUM VALUE FOR RESOURCE 3:2

ALLOCATED FROM RESOURCE 1:1

ALLOCATED FROM RESOURCE 2:0

ALLOCATED FROM RESOURCE 3:0

PROCESS 2

MAXIMUM VALUE FOR RESOURCE 1:6

MAXIMUM VALUE FOR RESOURCE 2:1

MAXIMUM VALUE FOR RESOURCE 3:3

ALLOCATED FROM RESOURCE 1:5

ALLOCATED FROM RESOURCE 2:1

ALLOCATED FROM RESOURCE 3:1

PROCESS 3

MAXIMUM VALUE FOR RESOURCE 1:3

MAXIMUM VALUE FOR RESOURCE 2:1

MAXIMUM VALUE FOR RESOURCE 3:4

ALLOCATED FROM RESOURCE 1:2

Reg.No:

ALLOCATED FROM RESOURCE 2:1

ALLOCATED FROM RESOURCE 3:1

PROCESS 4

MAXIMUM VALUE FOR RESOURCE 1:4

MAXIMUM VALUE FOR RESOURCE 2:2

MAXIMUM VALUE FOR RESOURCE 3:2

ALLOCATED FROM RESOURCE 1:0

ALLOCATED FROM RESOURCE 2:0

ALLOCATED FROM RESOURCE 3:2

ENTER TOTAL VALUE OF RESOURCE 1:9

ENTER TOTAL VALUE OF RESOURCE 2:3

ENTER TOTAL VALUE OF RESOURCE 3:6

OUTPUT:

RESOURCES	ALLOCATED	NEEDED	TOTAL	AVAIL
P1	322	100	222	936
P2	613	511	102	112
P3	314	211	103	
P4	422	002	420	

AVAIL BEFORE AVAIL AFTER

P 2 010 623

P 1 401 723

Reg.No:

P 3	620	934
P 4	514	936

THE ABOVE SEQUENCE IS A SAFE SEQUENCE

INPUT:

//TEST CASE:2

ENTER THE NO. OF PROCESSES:4

ENTER THE NO. OF RESOURCES:3

PROCESS 1

MAXIMUM VALUE FOR RESOURCE 1:3

MAXIMUM VALUE FOR RESOURCE 2:2

MAXIMUM VALUE FOR RESOURCE 3:2

ALLOCATED FROM RESOURCE 1:1

ALLOCATED FROM RESOURCE 2:0

ALLOCATED FROM RESOURCE 3:1

PROCESS 2

MAXIMUM VALUE FOR RESOURCE 1:6

MAXIMUM VALUE FOR RESOURCE 2:1

MAXIMUM VALUE FOR RESOURCE 3:3

ALLOCATED FROM RESOURCE 1:5

ALLOCATED FROM RESOURCE 2:1

Reg.No:

ALLOCATED FROM RESOURCE 3:1

PROCESS 3

MAXIMUM VALUE FOR RESOURCE 1:3

MAXIMUM VALUE FOR RESOURCE 2:1

MAXIMUM VALUE FOR RESOURCE 3:4

ALLOCATED FROM RESOURCE 1:2

ALLOCATED FROM RESOURCE 2:1

ALLOCATED FROM RESOURCE 3:2

PROCESS 4

MAXIMUM VALUE FOR RESOURCE 1:4

MAXIMUM VALUE FOR RESOURCE 2:2

MAXIMUM VALUE FOR RESOURCE 3:2

ALLOCATED FROM RESOURCE 1:0

ALLOCATED FROM RESOURCE 2:0

ALLOCATED FROM RESOURCE 3:2

ENTER TOTAL VALUE OF RESOURCE 1:9

ENTER TOTAL VALUE OF RESOURCE 2:3

ENTER TOTAL VALUE OF RESOURCE 3:6

Reg.No:

	RESOURCES	ALLOCATED	NEEDED	TOTAL	AVAIL
P1	322	101	221	936	110
P2	613	511	102		
P3	314	212	102		
P4	422	002	420		

OUTPUT:

AVAIL BEFORE AVAIL AFTER

DEADLOCK OCCURED

Experiment –4 :

AIM: Write a program to implement the producer-consumer problem using semaphores.

Program:

```
#include<stdio.h>
#include<stdlib.h>

int mutex=1,full=0,empty=3,x=0;

int main()
{
    int n;
    void producer();
    void consumer();
    int wait(int);
    int signal(int);
    printf("\n1.Producer\n2.Consumer\n3.Exit");
    while(1)
    {
        printf("\nEnter your choice:");
        scanf("%d",&n);
        switch(n)
        {
            case 1:  if((mutex==1)&&(empty!=0))
                      producer();
                    else
                      printf("Buffer is full!!");
                    break;
            case 2:  if((mutex==1)&&(full!=0))
                      consumer();
                    else
                      printf("Buffer is empty!!");
                    break;
            case 3:
                      exit(0);
                      break;
        }
    }
}
```

Reg.No:

```
    return 0;
}

int wait(int s)
{
    return (--s);
}

int signal(int s)
{
    return(++s);
}

void producer()
{
    mutex=wait(mutex);
    full=signal(full);
    empty=wait(empty);
    x++;
    printf("\nProducer produces the item %d",x);
    mutex=signal(mutex);
}

void consumer()
{
    mutex=wait(mutex);
    full=wait(full);
    empty=signal(empty);
    printf("\nConsumer consumes item %d",x);
    x--;
    mutex=signal(mutex);
}
```


Reg.No:

Output:

```
(cloudera@quickstart Desktop)$ cc pc.c -o pc
(cloudera@quickstart Desktop)$ ./pc

1.Producer
2.Consumer
3.Exit
Enter your choice:1

Producer produces the item 1
Enter your choice:2

Consumer consumes item 1
Enter your choice:1

Producer produces the item 1
Enter your choice:2

Consumer consumes item 1
Enter your choice:2
Buffer is empty!!
Enter your choice:1

Producer produces the item 1
Enter your choice:1

Producer produces the item 2
Enter your choice:2

Consumer consumes item 2
Enter your choice:2

Consumer consumes item 1
Enter your choice:2
Buffer is empty!!
Enter your choice:1

Producer produces the item 1
Enter your choice:1

Producer produces the item 2
Enter your choice:1

Producer produces the item 3
Enter your choice:1
Buffer is full!!
Enter your choice:3
(cloudera@quickstart Desktop)$
```

EXPERIMENT - 5:

AIM: Write a program to implement IPC using shared memory.

Shared Memory is a type of IPC where the two processes share same memory chunk and use it for IPC. One process writes into that memory and other reads it.

After running the Server you can see the attached Shared Memory

```
vgupta80@linux unixprog> ipcs -m
```

```
----- Shared Memory Segments -----
key      shmid  owner    perms   bytes   nattch   status
0x0000162e 65537   vgupta80 666     27      1
```

After running the client the memory is freed.

```
----- Shared Memory Segments -----
key      shmid  owner    perms   bytes   nattch   status
0x0000162e 65537   vgupta80 666     27      0
```

```
//SHMServer.C
```

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

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

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

```
#include <stdio.h>
```

```
#include <stdlib.h>
```

```
#define MAXSIZE 27
```

```
void die(char *s)
```

```
{
```

```
    perror(s);
```

```
    exit(1);
```

```
}
```

```
int main()
```

```
{
```

```
    char c;
```

```
    int shmid;
```

```
    key_t key;
```

```
    char *shm, *s;
```

```
    key = 5678;
```

```
    if ((shmid = shmget(key, MAXSIZE, IPC_CREAT | 0666)) < 0)
```

Reg.No:

```
    die("shmget");

if ((shm = shmat(shmid, NULL, 0)) == (char *) -1)
    die("shmat");

/*
 *   * Put some things into the memory for the
 *   other process to read.
 *   */
s = shm;

for (c = 'a'; c <= 'z'; c++)
    *s++ = c;

/*
 * Wait until the other process
 * changes the first character of our memory
 * to '*', indicating that it has read what
 * we put there.
 */
while (*shm != '*')
    sleep(1);

exit(0);
}
```

//SHMClient.C

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <stdio.h>
#include <stdlib.h>
#define MAXSIZE 27

void die(char *s)
{
    perror(s);
    exit(1);
}

int main()
{
    int shmid;
    key_t key;
```

Reg.No:

```
char *shm, *s;

key = 5678;

if ((shm = shmget(key, MAXSIZE, 0666)) < 0)
    die("shmget");

if ((shm = shmat(shmid, NULL, 0)) == (char *) -1)
    die("shmat");

//Now read what the server put in the memory.
for (s = shm; *s != '\0'; s++)
    putchar(*s);
putchar('\n');

/*
 *Change the first character of the
 *segment to '*', indicating we have read
 *the segment.
 */
*shm = '*';

exit(0);
}
```

EXPERIMENT - 6:

AIM: Write a C program to simulate page replacement algorithms

- a) FIFO b) LRU c) LFU

a) FIFO PAGE REPLACEMENT ALGORITHM

PROGRAM:

```
#include<stdio.h>

main()
{
    int i, j, k, f, pf=0, count=0, rs[25], m[10], n;

    printf("\n Enter the length of reference string -- ");

    scanf("%d",&n);

    printf("\n Enter the reference string -- ");

    for(i=0;i<n;i++)

        scanf("%d",&rs[i]);

    printf("\n Enter no. of frames -- ");

    scanf("%d",&f);

    for(i=0;i<f;i++)

        m[i]=-1;

    printf("\n The Page Replacement Process is -- \n");

    for(i=0;i<n;i++)

    {

        for(k=0;k<f;k++)

            {
```

Reg.No:

```
        if(m[k]==rs[i])
            break;
    }
    if(k==f)
    {
        m[count++]=rs[i];
        pf++;
    }
    for(j=0;j<f;j++)
        printf("\t%d",m[j]);
        if(k==f)
            printf("\tPF No. %d",pf);
        printf("\n");
        if(count==f)
            count=0;
    }
    printf("\n The number of Page Faults using FIFO are %d",pf);
}
```

INPUT

Enter the length of reference string – 20

Enter the reference string -- 7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1

Enter no. of frames – 3

OUTPUT

The Page Replacement Process is –

7	-1	-1	PF No. 1
7	0	-1	PF No. 2
7	0	1	PF No. 3
2	0	1	PF No. 4
2	0	1	
2	3	1	PF No. 5
2	3	0	PF No. 6
4	3	0	PF No. 7
4	2	0	PF No. 8
4	2	3	PF No. 9
0	2	3	PF No. 10
0	2	3	
0	2	3	
0	1	3	PF No. 11
0	1	2	PF No. 12
0	1	2	
0	1	2	
7	1	2	PF No. 13
7	0	2	PF No. 14
7	0	1	PF No. 15

The number of Page Faults using FIFO are 15

b.) LRU PAGE REPLACEMENT ALGORITHM

PROGRAM:

```
#include<stdio.h>

#include<conio.h>

main()

{

inti, j , k, min, rs[25], m[10], count[10], flag[25], n, f, pf=0, next=1;

clrscr();

printf("Enter the length of reference string -- ");

scanf("%d",&n);

printf("Enter the reference string -- ");

for(i=0;i<n;i++)

{

scanf("%d",&rs[i]);

flag[i]=0;

}

printf("Enter the number of frames -- ");

scanf("%d",&f);

for(i=0;i<f;i++)

{

count[i]=0;

m[i]=-1;

}
```


Reg.No:

```
printf("\nThe Page Replacement process is -- \n");
```

```
for(i=0;i<n;i++)
```

```
{
```

```
for(j=0;j<f;j++)
```

```
{
```

```
if(m[j]==rs[i])
```

```
{
```

```
flag[i]=1;
```

```
count[j]=next;
```

```
next++;
```

```
}
```

```
}
```

```
if(flag[i]==0)
```

```
{
```

```
if(i<f)
```

```
{
```

```
m[i]=rs[i];
```

```
count[i]=next;
```

```
next++;
```

```
}
```

```
else
```

```
{
```

```
min=0;
```

Reg.No:

```
for(j=1;j<f;j++)
if(count[min] > count[j])
min=j;
m[min]=rs[i];
count[min]=next;
next++;
}
pf++;
}
for(j=0;j<f;j++)
printf("%d\t", m[j]);
if(flag[i]==0)
printf("PF No. -- %d" ,pf);
printf("\n");
}
printf("\nThe number of page faults using LRU are %d",pf);
getch();
}
```

INPUT

Enter the length of reference string -- 20

Enter the reference string -- 7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1

Enter the number of frames – 3

OUTPUT

The Page Replacement process is --

7	-1	-1	PF No. -- 1
7	0	-1	PF No. -- 2
7	0	1	PF No. -- 3
2	0	1	PF No. -- 4
2	0	1	
2	0	3	PF No. -- 5
2	0	3	
4	0	3	PF No. -- 6
4	0	2	PF No. -- 7
4	3	2	PF No. -- 8
0	3	2	PF No. -- 9
0	3	2	
0	3	2	
1	3	2	PF No. -- 10
1	3	2	
1	0	2	PF No. -- 11
1	0	2	
1	0	7	PF No. -- 12
1	0	7	
1	0	7	

The number of page faults using LRU are 12

c.) LFU PAGE REPLACEMENT ALGORITHM

PROGRAM:

```
#include<stdio.h>

main()
{
    intrs[50], i, j, k, m, f, cntr[20], a[20], min, pf=0;

    printf("\nEnter number of page references -- ");

    scanf("%d",&m);

    printf("\nEnter the reference string -- ");

    for(i=0;i<m;i++)

        scanf("%d",&rs[i]);

    printf("\nEnter the available no. of frames -- ");

    scanf("%d",&f);

    for(i=0;i<f;i++)

    {

        cntr[i]=0;

        a[i]=-1;

    }

    printf("\nThe Page Replacement Process is – \n");

    for(i=0;i<m;i++)

    {

        for(j=0;j<f;j++)

            if(rs[i]==a[j])
```

Reg.No:

```
{  
cntr[j]++;  
break;  
}  
if(j==f)  
{  
min = 0;  
for(k=1;k<f;k++)  
if(cntr[k]<cntr[min])  
min=k;  
a[min]=rs[i];  
cntr[min]=1;  
pf++;  
}  
printf("\n");  
for(j=0;j<f;j++)  
printf("\t%d",a[j]);  
if(j==f)  
printf("\tPF No. %d",pf);  
}  
printf("\n\n Total number of page faults -- %d",pf);  
}
```

Reg.No:

INPUT:

Enter number of page references --10

Enter the reference string -- 1 2 3 4 5 2 5 2 5 1 4 3

Enter the available no. of frames – 3

OUTPUT:

The Page Replacement Process is –

1	-1	-1	PF No. 1
---	----	----	----------

1	2	-1	PF No. 2
---	---	----	----------

1	2	3	PF No. 3
---	---	---	----------

4	2	3	PF No. 4
---	---	---	----------

5	2	3	PF No. 5
---	---	---	----------

5	2	3	
---	---	---	--

5	2	3	
---	---	---	--

5	2	1	PF No. 6
---	---	---	----------

5	2	4	PF No. 7
---	---	---	----------

5	2	3	PF No. 8
---	---	---	----------

Total number of page faults -- 8

EXPERIMENT - 7:

AIM : Write a program to simulate the following contiguous memory allocation techniques

a) Worst-fit b) Best-fit c) First-fit

a) Worst Fit

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
#define max 25
void main()
{
int frag[max],b[max],f[max],i,j,nb,nf,temp;
static int bf[max],ff[max];
clrscr();
printf("\n\tMemory Management Scheme - First Fit");
printf("\nEnter the number of blocks:");
scanf("%d",&nb);
printf("Enter the number of files:");
scanf("%d",&nf);
printf("\nEnter the size of the blocks:-\n");
for(i=1;i<=nb;i++)
{
printf("Block %d:",i);
scanf("%d",&b[i]);
}
printf("Enter the size of the files :-\n");
for(i=1;i<=nf;i++)
{
printf("File %d:",i);
scanf("%d",&f[i]);
}
for(i=1;i<=nf;i++)
{
for(j=1;j<=nb;j++)
{
if(bf[j]!=1)
{
temp=b[j]-f[i];
if(temp>=0)
{
ff[i]=j;
break;
}
}
}
}
```

Reg.No:

```
frag[i]=temp;
bf[ff[i]]=1;
}
printf("\nFile_no:\tFile_size :\tBlock_no:\tBlock_size:\tFragement");
for(i=1;i<=nf;i++)
printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d",i,f[i],ff[i],b[ff[i]],frag[i]);
getch();
}
```

INPUT

Enter the number of blocks: 3

Enter the number of files: 2

Enter the size of the blocks:-

Block 1: 5

Block 2: 2

Block 3: 7

Enter the size of the files:-

File 1: 1

File 2: 4

OUTPUT

File No	File Size	Block No	Block Size	Fragment
1	1	1	5	4
2	4	3	7	3

b) Best Fit

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
#define max 25
void main()
{
int frag[max],b[max],f[max],i,j,nb,nf,temp,lowest=10000;
static int bf[max],ff[max];
clrscr();
printf("\nEnter the number of blocks:");
scanf("%d",&nb);
printf("Enter the number of files:");
scanf("%d",&nf);
printf("\nEnter the size of the blocks:-\n");
for(i=1;i<=nb;i++)
{
printf("Block %d:",i);
scanf("%d",&b[i]);
}
printf("Enter the size of the files :-\n");
for(i=1;i<=nf;i++)
{
printf("File %d:",i);
scanf("%d",&f[i]);
}
for(i=1;i<=nf;i++)
{
for(j=1;j<=nb;j++)
{
if(bf[j]!=1)
{
temp=b[j]-f[i];
if(temp>=0)
if(lowest>temp)
{
ff[i]=j;

lowest=temp;
}
}
}
}
frag[i]=lowest;
bf[ff[i]]=1;
lowest=10000;
}
```

Reg.No:

```
printf("\nFile No\tFile Size \tBlock No\tBlock Size\tFragment");
for(i=1;i<=nf && ff[i]!=0;i++)
printf("\n%d\t%d\t%d\t%d\t%d",i,f[i],ff[i],b[ff[i]],frag[i]);
getch();
}
```

INPUT

Enter the number of blocks: 3

Enter the number of files: 2

Enter the size of the blocks:-

Block 1: 5

Block 2: 2

Block 3: 7

Enter the size of the files:-

File 1: 1

File 2: 4

OUTPUT

File No	File Size	Block No	Block Size	Fragment
1	1	2	2	1
2	4	1	5	1

c) First Fit

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
#define max 25
void main()
{
int frag[max],b[max],f[max],i,j,nb,nf,temp,highest=0;
static int bf[max],ff[max];
clrscr();
printf("\n\tMemory Management Scheme - Worst Fit");
printf("\nEnter the number of blocks:");
scanf("%d",&nb);
printf("Enter the number of files:");
scanf("%d",&nf);
printf("\nEnter the size of the blocks:-\n");
for(i=1;i<=nb;i++)
{
printf("Block %d:",i);
scanf("%d",&b[i]);
}
printf("Enter the size of the files :-\n");
for(i=1;i<=nf;i++)
{
printf("File %d:",i);
scanf("%d",&f[i]);
}
for(i=1;i<=nf;i++)
{
for(j=1;j<=nb;j++)
{
if(bf[j]!=1) //if bf[j] is not allocated
{
temp=b[j]-f[i];
if(temp>=0)
if(highest<temp)
{
ff[i]=j;
highest=temp;
}
}
}
}
frag[i]=highest;
bf[ff[i]]=1;
highest=0;
```

Reg.No:

```
}  
printf("\nFile_no:\tFile_size :\tBlock_no:\tBlock_size:\tFragement");  
for(i=1;i<=nf;i++)  
printf("\n%d\t%d\t%d\t%d\t%d",i,f[i],ff[i],b[ff[i]],frag[i]);  
getch();  
}
```

INPUT

Enter the number of blocks: 3

Enter the number of files: 2

Enter the size of the blocks:-

Block 1: 5

Block 2: 2

Block 3: 7

Enter the size of the files:-

File 1: 1

File 2: 4

OUTPUT

File No	File Size	Block No	Block Size	Fragment
1	1	3	7	6
2	4	1	5	1

EXPERIMENT - 8:

Aim: Write a program to implement Paging technique for memory management.

Program:

```
#include<stdio.h>
#include<conio.h>
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
    {
```

Reg.No:

```
pa=fno[x][y]*ps+offset;
printf("\nThe Physical Address is -- %d",pa);
}
getch();
}
```

INPUT

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

OUTPUT

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

EXPERIMENT -9:

AIM :

Write a C program to simulate the following file allocation strategies.

a) Sequential b) Linked c)) Indexed

a)SEQUENTIAL FILE ALLOCATION :

PROGRAM :

```
#include<stdio.h>
struct fileTable
{
    char name[20];
    int sb, nob;
}ft[30];
void main()
{
    int i, j, n;
    char s[20];
    printf("Enter no of files :");
    scanf("%d",&n);
    for(i=0;i<n;i++)
    {
        printf("\nEnter file name %d :",i+1);
        scanf("%s",ft[i].name);
        printf("Enter starting block of file %d :",i+1);
        scanf("%d",&ft[i].sb);
        printf("Enter no of blocks in file %d :",i+1);
        scanf("%d",&ft[i].nob);
    }
    printf("\nEnter the file name to be searched-- ");
    scanf("%s",s);
    for(i=0;i<n;i++)
        if(strcmp(s, ft[i].name)==0)
            Break;
    if(i==n)
        printf("\nFile Not Found");
    else
```

Reg.No:

```
{
    printf("\nFILE NAME START BLOCK NO OF BLOCKS BLOCKS
    OCCUPIED\n");
    printf("\n%s\t\t%d\t\t%d\t\t",ft[i].name,ft[i].sb,ft[i].nob);
    for(j=0;j<ft[i].nob;j++)
        printf("%d, ",ft[i].sb+j);
}
}
```

INPUT :

Enter no of files :3
Enter file name 1 :A
Enter starting block of file 1 :85
Enter no of blocks in file 1 :6
Enter file name 2 :B
Enter starting block of file 2 :102
Enter no of blocks in file 2 :4
Enter file name 3 :C
Enter starting block of file 3 :60
Enter no of blocks in file 3 :4
Enter the file name to be searched -- B

OUTPUT :

FILE NAME	START BLOCK	NO OF BLOCKS	BLOCKS OCCUPIED
B	102	4	102, 103, 104, 105

b)LINKED FILE ALLOCATION :

PROGRAM :

```
#include<stdio.h>
>
struct fileTable
{
    char name[20];
    int nob;
    struct block *sb;
}ft[30];
struct block
{
    int bno;
    struct block *next;
};
void main()
{
    int i, j, n;
    char s[20];
    struct block *temp;
    printf("Enter no of files :");
    scanf("%d",&n);
    for(i=0;i<n;i++)
    {
        printf("\nEnter file name %d :",i+1);
        scanf("%s",ft[i].name);
        printf("Enter no of blocks in file %d :",i+1);
        scanf("%d",&ft[i].nob);
        ft[i].sb=(struct block*)malloc(sizeof(struct block));
        temp = ft[i].sb;
        printf("Enter the blocks of the file :");
        scanf("%d",&temp->bno);
        temp->next=NULL;
        for(j=1;j<ft[i].nob;j++)
        {
            temp->next = (struct block*)malloc(sizeof(struct block));
            temp = temp->next;
            scanf("%d",&temp->bno);
```

Reg.No:

```
        }
        temp->next = NULL;
    }
    printf("\nEnter the file name to be searched -- ");
    scanf("%s",s);
    for(i=0;i<n;i++)
        if(strcmp(s, ft[i].name)==0)
            break;
    if(i==n)
        printf("\nFile Not Found");
    else
    {
        printf("\nFILE NAME NO OF BLOCKS BLOCKS OCCUPIED");
        printf("\n %s\t\t%d\t",ft[i].name,ft[i].nob);
        temp=ft[i].sb;
        for(j=0;j<ft[i].nob;j++)
        {
            printf("%d -> ",temp->bno);
            temp = temp->next;
        }
    }
}
```

INPUT :

Enter no of files : 2

Enter file 1 : A

Enter no of blocks in file 1 : 4

Enter the blocks of the file 1 : 12 23 9 4

Enter file 2 : G

Enter no of blocks in file 2 : 5

Enter the blocks of the file 2 : 88 77 66 55 44

Enter the file to be searched : G

OUTPUT :

FILE NAME	NO OF BLOCKS	BLOCKS OCCUPIED
G	5	88-> 77-> 66-> 55-> 44

c)INDEXED FILE ALLOCATION :

PROGRAM :

```
#include<stdio.h>
struct fileTable
{
    char name[20];
    int nob, blocks[30];
}ft[30];
void main()
{
    int i, j, n;
    char s[20];
    printf("Enter no of files :");
    scanf("%d",&n);
    for(i=0;i<n;i++)
    {
        printf("\nEnter file name %d :",i+1);
        scanf("%s",ft[i].name);
        printf("Enter no of blocks in file %d :",i+1);
        scanf("%d",&ft[i].nob);
        printf("Enter the blocks of the file :");
        for(j=0;j<ft[i].nob;j++)
            scanf("%d",&ft[i].blocks[j]);
    }
    printf("\nEnter the file name to be searched-- ");
    scanf("%s",s);
    for(i=0;i<n;i++)
        if(strcmp(s, ft[i].name)==0)
            break;
    if(i==n)
        printf("\nFile Not Found");
    else
    {
        printf("\nFILE NAME NO OF BLOCKS BLOCKS OCCUPIED");
        printf("\n %s\t\t%d\t",ft[i].name,ft[i].nob);
        for(j=0;j<ft[i].nob;j++)
            printf("%d, ",ft[i].blocks[j]);
    }
}
```

Reg.No:

}

INPUT :

Enter no of files : 2

Enter file 1 : A

Enter no of blocks in file 1 : 4

Enter the blocks of the file 1 : 12 23 9 4

Enter file 2 : G

Enter no of blocks in file 2 : 5

Enter the blocks of the file 2 : 88 77 66 55 44

Enter the file to be searched : G

OUTPUT :

FILE NAME	NO OF BLOCKS	BLOCKS OCCUPIED
G	5	88, 77, 66, 55, 44

EXPERIMENT - 10:

Aim: Simulate the following Disk Scheduling Algorithms

(a) FCFS (b) SSTF (c) SCAN

a) FCFS Disk Scheduling

Program:

```
#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;
}
```

Output:-

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

b) SSTF Disk Scheduling

Program:

```
#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;
}
```

Reg.No:

Output:-

Enter the number of Request

8

Enter Request Sequence

95 180 34 119 11 123 62 64

Enter initial head Position

50

Total head movement is 236

c) **SCAN Disk Scheduling**

Program:

```
#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;
        }
    }
}
```


Reg.No:

```
// 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;
}
```

Reg.No:

Output:-

Enter the number of Request

8

Enter the Requests Sequence

95 180 34 119 11 123 62 64

Enter initial head position

50

Enter total disk size

200

Enter the head movement direction for high 1 and for low 0

1

Total head movement is 337