CYCLE-3	
S. No	**LIST OF EXPERIMENTS**
Exp-1	Write a program to solve producer-consumer problem using Semaphores.
Exp-2	Implement the following memory allocation methods for fixed partition. a) First fit b) Worst fit c) Best fit
Exp-3	Simulate the following page replacement algorithms a) FIFO b) LRU c) LFU
Exp-4	Simulate Paging Technique of memory management.

Experiment-1

1. Write a program to solve producer-consumer problem using Semaphores.

Program:

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <semaphore.h>
#include <pthread.h>

#define BUFFER_SIZE 5

// Shared variables
int buffer[BUFFER_SIZE];
int in = 0;
int out = 0;

// Semaphores
sem_t empty;
sem_t full;
sem_t mutex;

// Producer thread function
void *producer(void *arg) {
    int item;
    for (int i = 0; i < 10; i++) {
        item = rand() % 100;
        sem_wait(&empty); // Wait for empty slot
        sem_wait(&mutex); // Acquire lock

        buffer[in] = item;
        printf("Producer produced %d\n", item);
        in = (in + 1) % BUFFER_SIZE;
```

```
        sem_post(&mutex); // Release lock
        sem_post(&full); // Signal that buffer has an item
        sleep(1); // Simulate production time
    }
    return NULL;
}

// Consumer thread function
void *consumer(void *arg) {
    int item;
    for (int i = 0; i < 10; i++) {
        sem_wait(&full); // Wait for item in buffer
        sem_wait(&mutex); // Acquire lock

        item = buffer[out];
        printf("Consumer consumed %d\n", item);
        out = (out + 1) % BUFFER_SIZE;

        sem_post(&mutex); // Release lock
        sem_post(&empty); // Signal that buffer has an empty slot
        sleep(2); // Simulate consumption time
    }
    return NULL;
}

int main() {
    pthread_t tid_producer, tid_consumer;

    // Initialize semaphores
    sem_init(&empty, 0, BUFFER_SIZE); // Initially all slots are empty
    sem_init(&full, 0, 0); // Initially no items in buffer
    sem_init(&mutex, 0, 1); // Initially lock is available

    // Create producer and consumer threads
    pthread_create(&tid_producer, NULL, producer, NULL);
    pthread_create(&tid_consumer, NULL, consumer, NULL);

    // Wait for threads to finish
    pthread_join(tid_producer, NULL);
    pthread_join(tid_consumer, NULL);

    // Destroy semaphores
    sem_destroy(&empty);
    sem_destroy(&full);
    sem_destroy(&mutex);
```

```
    return 0;
}
```

Output:

```
cilab@ubuntu:~$ gcc -pthread pcsemaphore1.c
cilab@ubuntu:~$ ./a.out
Producer produced 83
Consumer consumed 83
Producer produced 86
Consumer consumed 86
Producer produced 77
Producer produced 15
Consumer consumed 77
Producer produced 93
Producer produced 35
Consumer consumed 15
Producer produced 86
Producer produced 92
Consumer consumed 93
Producer produced 49
Producer produced 21
Consumer consumed 35
Consumer consumed 86
Consumer consumed 92
Consumer consumed 49
Consumer consumed 21
```

Experiment-2

1. Implement the following memory allocation methods for fixed partitions.
a) First fit b) Worst fit c) Best fit

a) First fit

```
#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];
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;
}
printf("\nFile No\tFile Size \tBlock No\tBlock Size\tFragment");
for(i=1;i<=nf && ff[i]!=0;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]);
}
```

Output:

```
Enter the number of blocks:6
Enter the number of files:5

Enter the size of the blocks:-
Block 1:300
Block 2:600
Block 3:350
Block 4:200
Block 5:750
Block 6:125
Enter the size of the files :-
File 1:115
File 2:500
File 3:358
File 4:200
File 5:375

File No File Size       Block No       Block Size     Fragment
1              115            6              125            10
2              500            2              600            100
3              358            5              750            392
4              200            4              200            0
```

b) Worst fit

```
#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\t Memory Management Scheme - Worst Fit");
printf("\n Enter 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;
}
}
}
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();
}
```

Output:

```
cilab@ubuntu:~$ gcc -pthread worstfit.c
cilab@ubuntu:~$ ./a.out

Memory Management Scheme - Worst Fit
 Enter the number of blocks:6
Enter the number of files:5

Enter the size of the blocks:-
Block 1:300
Block 2:600
Block 3:350
Block 4:200
Block 5:750
Block 6:125
Enter the size of the files :-
File 1:115
File 2:500
File 3:358
File 4:200
File 5:375

File_no: File_size : Block_no: Block_size: Fragement
1            115            1            300            185
2            500            2            600            100
3            358            5            750            392
```

4 200 3 350 150
5

c) Best fit

```
#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("\n\t Memory Management Scheme - Best 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)
if(lowest>temp)
{
ff[i]=j;

lowest=temp;
}
}
}
frag[i]=lowest;
bf[ff[i]]=1;
lowest=10000;
}
printf("\nFile No\tFile Size \tBlock No\tBlock Size\tFragment");
for(i=1;i<=nf && ff[i]!=0;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();
}
```

Output:

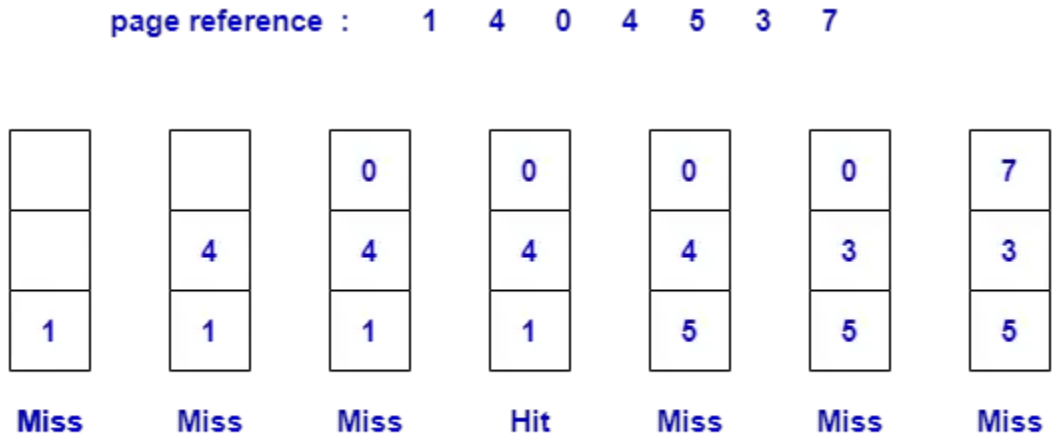
```
Enter the number of blocks:6
Enter the number of files:5

Enter the size of the blocks:-
Block 1:300
Block 2:600
Block 3:350
Block 4:200
Block 5:750
Block 6:125
Enter the size of the files :-
File 1:115
File 2:500
File 3:358
File 4:200
File 5:375

File No File Size       Block No        Block Size      Fragment
1               115             6               125             10
2               500             2               600             100
3               358             5               750             392
4               200             4               200             0
Process returned 0 (0x0)   execution time : 54.215 s
Press any key to continue.
```

Experiment-3

Simulate the following page replacement algorithms
a) FIFO b) LRU c) LFU

(a) FIFO

FIFO page-replacement algorithm is a technique in which a new page replaces the page which has been in the main memory for the longest time. This means the page which comes in the frame first will replace first.
In this algorithm, we will use a queue data structure.



When the page number is added in a frame and the page number is not present in a frame then a **page miss** will occur and if the page number is present in a frame then a **page hit** occurs.

Program:

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

int pagefault(int a[], int frame[], int n, int no) {
    int i, j, avail, count = 0, k;
    /* initialize frame with value -1 */
    for (i = 0; i < no; i++) {
        frame[i] = -1;
    }
    j = 0;
    for (i = 0; i < n; i++) {
        avail = 0;
        for (k = 0; k < no; k++)
        /* if equal it means page number(reference) is available in frame */
            if (frame[k] == a[i])
                avail = 1;
        /* if avail=0 means page is not available in frame */
        if (avail == 0) {
            frame[j] = a[i];
            /* j will calculate the position at which the new page add */
            j = (j + 1) % no;
            count++;  // variable count calculates the total page fault
        }
    }
    return count;
}
void main() {
    int n, i, * a, * frame, no, fault;
    printf("\nENTER THE NUMBER OF PAGES:\n");
    scanf("%d", & n);

    a = (int * ) malloc(n * sizeof(int));
    printf("ENTER THE PAGE NUMBER :\n");
    for (i = 0; i < n; i++)
        scanf("%d", & a[i]);

    printf("ENTER THE NUMBER OF FRAMES :");
    scanf("%d", & no);

    frame = (int * ) malloc(no * sizeof(int));
    fault = pagefault(a, frame, n, no);
    printf("Page Fault Is %d", fault);
}
```

Output:

```
ENTER THE NUMBER OF PAGES:
7
ENTER THE PAGE NUMBER :
1       4       0       4       5       3       7
```

```
ENTER THE NUMBER OF FRAMES :3
Page Fault Is 6

ENTER THE NUMBER OF PAGES:
20
ENTER THE PAGE NUMBER :
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
Page Fault Is 15
```

(b) LRU

```
//LRU page replacement algorithm program in C
#include<stdio.h>
#include <stdlib.h>

void reframe(int * frame, int frames, int index) {
    /* send the current index to the end of the array */
    int k, temp;
    for (k = index; k < frames - 1; k++) { //iterate frame
        temp = * (frame + k);
        *(frame + k) = * (frame + k + 1);
        *(frame + k + 1) = temp;
        //printf("%d ",*(frame+k));
    }
    //printf("\n");
}

int lru(int csno, int * controlstring, int frames) {
    int pf = 1, * frame, i, j = 0, k, avail = 0, index;
    frame = calloc(sizeof(int), frames);
    for (i = 0; i < csno; i++) {
        /*iterate cntro string */
        avail = 0;
        for (k = 0; k < frames; k++) { //iterate frame
            printf("%d ", *(frame + k));
            if ( * (frame + k) == * (controlstring + i)) {
                avail = 1;
                index = k;
            }
```

```
        }
        // printf("\n");
        if (avail == 1) {
            /*if element available in frame */
            reframe(frame, frames, index);
        } else {
            /*element not available in frame */
            *(frame + 0) = * (controlstring + i);
            reframe(frame, frames, 0);
            pf++;
        }
        printf("\n");
    }
    for (k = 0; k < frames; k++)
        printf("%d ", *(frame + k));
    return pf;
}
int main() {
    int csno, i, * controlstring, frames, pf;
    printf("enter the no of pages in control strings:");
    scanf("%d", & csno);

    controlstring = malloc(sizeof(int) * csno);

    printf("enter the control strings:");
    for (i = 0; i < csno; i++) {
        scanf("%d", controlstring + i);
    }

    printf("enter the no of frames:");
    scanf("%d", & frames);

    pf = lru(csno, controlstring, frames);

    printf("no of page fault is %d", pf);
    return 0;
}
```

Output:

```
enter the no of pages in control strings:20
enter the control strings:7     0       1       2       0       3       0       4       2       3
0 3       2       1       2       0       1       7       0       1
enter the no of frames:4
0 0 0 0
0 0 0 7
0 0 7 0
0 7 0 1
7 0 1 2
7 1 2 0
1 2 0 3
1 2 3 0
2 3 0 4
3 0 4 2
```

0 4 2 3
4 2 3 0
4 2 0 3
4 0 3 2
0 3 2 1
0 3 1 2
3 1 2 0
3 2 0 1
2 0 1 7
2 1 7 0
2 7 0 1 no of page fault is 8

(c) LFU

```
#include<stdio.h>
void print(int frameno,int frame[])
{
        int j;
        for(j=0;j<frameno;j++)
        printf("%d\t",frame[j]);
        printf("\n");
}
int main()
{
        int i,j,k,n,page[50],frameno,frame[10],move=0,flag,count=0,count1[10]={0},
                repindex,leastcount;
        float rate;
        printf("Enter the number of pages\n");
        scanf("%d",&n);
        printf("Enter the page reference numbers\n");
        for(i=0;i<n;i++)
        scanf("%d",&page[i]);
        printf("Enter the number of frames\n");
        scanf("%d",&frameno);
        for(i=0;i<frameno;i++)
        frame[i]=-1;
        printf("Page reference string\tFrames\n");
        for(i=0;i<n;i++)
        {
                printf("%d\t\t\t",page[i]);
                flag=0;
                for(j=0;j<frameno;j++)
```

```
                {
                        if(page[i]==frame[j])
                        {
                                flag=1;
                                count1[j]++;
                                printf("No replacement\n");
                                break;
                        }
                }
                if(flag==0&&count<frameno)
                {
                        frame[move]=page[i];
                        count1[move]=1;
                        move=(move+1)%frameno;
                        count++;
                        print(frameno,frame);
                }
                else if(flag==0)
                {
                        repindex=0;
                        leastcount=count1[0];
                        for(j=1;j<frameno;j++)
                        {
                                if(count1[j]<leastcount)
                                {
                                        repindex=j;
                                        leastcount=count1[j];
                                }
                        }

                        frame[repindex]=page[i];
                        count1[repindex]=1;
                        count++;
                        print(frameno,frame);
                }
        }
        rate=(float)count/(float)n;
        printf("Number of page faults is %d\n",count);
        printf("Fault rate is %f\n",rate);
        return 0;
}
```

Output:

```
Enter the number of pages
12
Enter the page reference numbers
0       2       1       6       4       0       1       0       3       1       2       1
Enter the number of frames
4
Page reference string   Frames
0                       0       -1      -1      -1
2                       0       2       -1      -1
1                       0       2       1       -1
```

```
6                   0       2       1       6
4                   4       2       1       6
0                   0       2       1       6
1                   No replacement
0                   No replacement
3                   0       3       1       6
1                   No replacement
2                   0       2       1       6
1                   No replacement
Number of page faults is 8
Fault rate is 0.666667
```

Experiment-4:

1. Simulate Paging Technique of memory management.

```
#include<stdio.h>
#include<conio.h>
void main()
{
 int ms, ps, nop, np, rempages, i, j, x, y, pa, offset;
 int s[10], fno[10][20];

//clrscr();

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 50

The Physical Address is -- 750