School of Computer Engineering

Department of BCA

Lab Manual and Student Guide

Operating System Lab

Course	**BCA**	**Name of Lab**	**Operating System**
Session	**2021-22**	**Subject Code**	BCACCA2203
Year	**1nd**	**Faculty**	**Mr. Gaurav Kr Das**
Semester	**2th**	**Lab Assistant**	

TABLE OF CONTENTS

DO'S AND DON'TS

DO'S:

- Students must be in proper uniform with their I-Cards.
- Student should fill the entry register.
- Shut down the computers before leaving the lab.
- Keep the bags in the racks.
- Enter and leave the lab on proper time.
- Maintain the decorum of the lab.
- Utilize lab hours in the corresponding experiment.
- Students should bring their respective lab records.
- Students should get permission from Technical Assistant before taking out any kind of data.
- Students require checking their Lab Records by Faculty on Time and must fill the Evaluation sheet.

DON'TS

Don't bring any external material in the lab.

2. Don't make noise in the lab

3. Don't bring the mobile in the lab. If extremely necessary then keep in silent mode.

4. Don't enter in server room without permission of lab in-charge.

5. Don't delete or make any modification in system files.

7. Don't carry any lab equipments outside the lab.

INSTRUCTIONS

Before entering in the lab

- All the students are supposed to prepare the theory regarding the next experiment.
- Students are supposed to bring the practical file and the lab copy.
- Previous programs should be written in the practical file.
- All the students must follow the instructions, failing which he/she may not be allowed in the lab.

While working in the lab

- Adhere to experimental schedule as instructed by the lab in-charge.
- Get the previously executed program signed by the instructor.
- Get the output of the current program checked by the instructor in the lab copy.
- Each student should work on his/her assigned computer at each turn of the lab.
- Take responsibility of valuable accessories.
- Concentrate on the assigned practical and do not play games
- If anyone caught red handed carrying any equipment of the lab, then he/she will have to face serious consequences.

SYLLABUS

Code: BCACCA2203 OPERATING SYSTEM LAB 1 Credits [LTP:0-0-2]

COURSE OUTCOME

Students will be able:

- To explain the difference between hardware, software; operating systems, programs and files
- To simulate the working of CPU scheduling algorithms
- To simulate the working of contiguous memory allocation problems

Sr. No.	Experiments
1	Create CPU Scheduling Algorithms For First Come First Serve(Fcfs)
2	Create CPU Scheduling Algorithms For shortest job first (SJF)
3	Create CPU Scheduling Algorithms For round robin algorithm
4	Create CPU Scheduling Algorithms For priority
5	Contiguous Memory Allocation On Worst Fit
6	Contiguous Memory Allocation On best Fit
7	Contiguous Memory Allocation On first Fit
8	Page Replacement Algorithm ON FIRST IN FIRST OUT (FIFO)
9	Page Replacement Algorithm ON least recently used (LRU)
10	Page Replacement Algorithm ON optimal

MARKS SCHEME

Examination Marks Scheme

Practical (Laboratory) Subjects:-

a. Continuous Internal Evaluation (CIE)	**40 marks**	**40%**
- **CIE-I (Pr.):** Performance, Lab Record, Viva, Attendance and Discipline	20 marks	20%
- **CIE-II(Pr.):** Performance, Lab Record, Viva, Attendance and Discipline	20 marks	20%
b. Mid Semester Exam (MSE)– One	**20** marks	**20 %**
c. End Semester Exam (ESE) – One	**40** marks	**40%**
TOTAL	**100**	**100 %**

LAB PLAN

Total number of experiment 12

Total number of turns required 12

Number of turns required for

Experiment Number	Turns Required	Turn No.
Exp. 1	1	1
Exp. 2	1	2
Exp. 3	1	3
Exp. 4	1	4
Exp. 5	1	5
Exp. 6	1	6
Exp. 7	1	7
Exp. 8	1	8
Exp. 9	1	9
Exp. 10	1	10
Exp. 11	1	11
Exp. 12	1	12

Distribution of lab hours

Attendance	05 minutes
Explanation of features of language	15 minutes
Explanation of experiment	15 minutes
Performance of experiment	70 minutes
Viva / Quiz / Queries	15 minutes

Total 120 minutes

LAB OBJECTIVE

Upon successful completion of this Lab the student will be able to: get the clear understanding of

1) CPU Scheduling algorithms 2) Memory Management 3) Unix Commands

List of Lab Exercises

Sr. No.	Name of experiment
	Cycle-1
1.	Create CPU Scheduling Algorithms For First Come First Serve(Fcfs).
2.	Create CPU Scheduling Algorithms For shortest job first (SJF)
3.	Create CPU Scheduling Algorithms For round robin algorithm.
4.	Create CPU Scheduling Algorithms For priority
	Cycle-2
5.	Contiguous Memory Allocation On Worst Fit
6.	Contiguous Memory Allocation On best Fit
7.	Contiguous Memory Allocation On first Fit
8.	Page Replacement Algorithm ON FIRST IN FIRST OUT (FIFO)
9.	Page Replacement Algorithm ON least recently used (LRU)
10.	Page Replacement Algorithm ON optimal
11.	Basic Linux Commands
12.	Use Of VI editor in Linux

Resources for all the labs

Hardware :

1.Computer &

2.Peripheral devices

Software:

1. Turbo C++

Text Books:

1. Operation System Concepts Tanenbaum 2nd Edition, Pearson Education

2. Operating System William Stallings 4th Edition, Pearson Education.

Reference Books:

1. Guide to UNIX Using LINUX Jack Dent Tony Gaddis, Vikas Thomson Pub. House Pvt. Ltd. 2010

Reference Websites:

1. www.tutorialpoint.com

Experiment-1

OBJECTIVE

Write a C program to simulate the following non-preemptive CPU scheduling algorithms to find turnaround time and waiting time for the First Come First Serve Scheduling Algorithm.

DESCRIPTION

Assume all the processes arrive at the same time.

FCFS CPU SCHEDULING ALGORITHM

For FCFS scheduling algorithm, read the number of processes/jobs in the system, their CPU burst times. The scheduling is performed on the basis of arrival time of the processes irrespective of their other parameters. Each process will be executed according to its arrival time. Calculate the waiting time and turnaround time of each of the processes accordingly.

First-Come, First-Served (FCFS) Scheduling

Process	Burst Time
P1	24
P2	**3**
P3	3

Suppose that the processes arrive in the order: P1 , P2 , P3 The Gantt Chart for the schedule is:

Waiting time for P1 = 0; P2 = 24; P3 = 27

Average waiting time: (0 + 4 +27)/3 = 17

ALGORITHM

1. Start
2. Declare the array size

3. Read the number of processes to be inserted

4. Read the Burst times of processes

5. calculate the waiting time
   of each process
   wt[i+1]=bt[i]+wt[i]
6. calculate the turnaround time
   of each process
   tt[i+1]=tt[i]+bt[i+1]
7. Calculate the average waiting time and average turnaround time.

8. Display the values

9. Stop

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int i,j,bt[10],n,wt[10],tt[10],w1=0,t1=0; float aw,at;
    clrscr();
    printf("enter no. of processes:\n"); scanf("%d",&n);
    printf("enter the burst time of processes:"); for(i=0;i<n;i++)
    scanf("%d",&bt[i]); for(i=0;i<n;i++)
    {
        wt[0]=0;
        tt[0]=bt[0];
        wt[i+1]=bt[i]+wt[i];
        tt[i+1]=tt[i]+bt[i+1];
        w1=w1+wt[i]; t1=t1+tt[i];
    }
        aw=w1/n; at=t1/n;
        printf("\nbt\t wt\t tt\n"); for(i=0;i<n;i++)
    printf("%d\t %d\t %d\n",bt[i],wt[i],tt[i]);
    printf("aw=%f\n,at=%f\n",aw,at); getch();
}
```

INPUT

Enter no of processes 3

enter bursttime 12 8 20

EXPECTED OUTPUT

bt wt tt 12 0 12 8 12 20 20 20 40 aw=10.666670 at=24.00000

VIVA QUESTIONS

1 *What is First-Come-First-Served (FCFS) Scheduling?*

2 Why CPU scheduling is required?

3 Which technique was introduced because a single job could not keep both the CPU and the I/O devices busy?

1)Time-sharing 2) SPOOLing 3) Preemptive scheduling 4) Multiprogramming

4 CPU performance is measured through___.

1)Throughput 2) MHz 3) Flaps 4) None of the above

5 Which of the following is a criterion to evaluate a scheduling algorithm?

1 CPU Utilization: Keep CPU utilization as high as possible.

2 Throughput: number of processes completed per unit time.

3 Waiting Time: Amount of time spent ready to run but not running.

4 All of the above

EXPERIMENT : 2

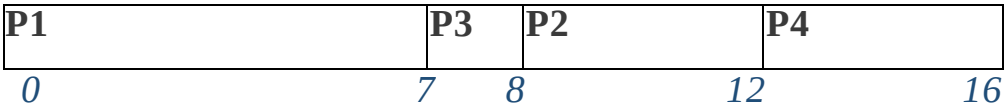
OBJECTIVE

Write a C program to simulate the following non-preemptive CPU scheduling algorithms to find turnaround time and waiting time for the
Shortest Job First Scheduling Algorithm.

THEORY:

Example of Non Preemptive SJF

Process	Arrival Time	Burst Time
P_1	0.0	7
P_2	2.0	4
P_3	4.0	1
P_4	3.0	4



ALGORITHM

Start

1 Declare the array size
2 Read the number of processes to be inserted
3 Read the Burst times of processes
4 Sort the Burst times in ascending order and process with shortest burst time is first executed.
5 Calculate the waiting time of each process wt[i+1]=bt[i]+wt[i]
6 Calculate the turnaround time of each process tt[i+1]=tt[i]+bt[i+1]
7 Calculate the average waiting time and average turnaround time.
8 Display the values
9 Stop

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
void main()
{
        int i,j,bt[10],t,n,wt[10],tt[10],w1=0,t1=0; float aw,at;
        clrscr();
        printf("enter no. of processes:\n"); scanf("%d",&n);
        printf("enter the burst time of processes:"); for(i=0;i<n;i++)
        scanf("%d",&bt[i]); for(i=0;i<n;i++)
 {
       for(j=i;j<n;j++) if(bt[i]>bt[j])
        {
             t=bt[i]; bt[i]=bt[j]; bt[j]=t;
         }
  }
    for(i=0;i<n;i++)
    printf("%d",bt[i]);
      for(i=0;i<n;i++)
           {
              wt[0]=0;
              tt[0]=bt[0];
              wt[i+1]=bt[i]+wt[i];
              tt[i+1]=tt[i]+bt[i+1];
             w1=w1+wt[i]; t1=t1+tt[i];
                }
              aw=w1/n;
              at=t1/n;
```

```
printf("\nbt\t wt\t tt\n"); for(i=0;i<n;i++)
printf("%d\t %d\t %d\n",bt[i],wt[i],tt[i]); printf("aw=%f\n,at=%f\n",aw,at);
getch();
}
```

INPUT:

enter no of processes 3

enter burst time 12 8 20

OUTPUT:

bt wt tt

12 8 20

8 0 8

20 20 40

aw=9.33 at=22.64

VIVA QUESTIONS:

The optimum CPU scheduling algorithm is
(A)FIFO (B)SJF with preemption. (C)SJF without preemption.(D)Round Robin.

In terms of average wait time the optimum scheduling algorithm is
(A)FCFS (B)SJF (C)Priority (D)RR

What are the dis-advantages of SJF Scheduling Algorithm?

What are the advantages of SJF Scheduling Algorithm?

Define CPU Scheduling algorithm?

EXPERIMENT : 3

Object: Simulate the following CPU Scheduling Algorithms

Round Robin

THEORY:

Round Robin:

Example of RR with time quantum=3

Process	**Burst time**
aaa	4
Bbb	3
Ccc	2
Ddd	5
Eee	1

ALGORITHM

1. Start
2. Declare the array size
3. Read the number of processes to be inserted
4. Read the burst times of the processes
5. Read the Time Quantum
6. if the burst time of a process is greater than time Quantum then subtract time quantum form the burst time Else

   Assign the burst time to time quantum.

   calculate the average waiting time and turn around time of the processes.
7. Display the values
8. Stop

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
void main()
{
int st[10],bt[10],wt[10],tat[10],n,tq;
int i,count=0,swt=0,stat=0,temp,sq=0;
float awt=0.0,atat=0.0;
clrscr();
printf("Enter number of processes:");
scanf("%d",&n);
printf("Enter burst time for sequences:");
for(i=0;i<n;i++)
{
scanf("%d",&bt[i]); st[i]=bt[i];
}
printf("Enter time quantum:"); scanf("%d",&tq);
while(1)
{
for(i=0,count=0;i<n;i++)
{
temp=tq;
if(st[i]==0)
{
count++;
continue;
}
```

```
            if(st[i]>tq)
             st[i]=st[i]-tq;
                  else if(st[i]>=0)
                {
                  temp=st[i]; st[i]=0;
                 }
                 sq=sq+temp; tat[i]=sq;
                }
              if(n==count) break;
        }
 for(i=0;i<n;i++)
  {
        wt[i]=tat[i]-bt[i];
        swt=swt+wt[i];
        stat=stat+tat[i];
       }
         awt=(float)swt/n;
        atat=(float)stat/n;
       printf("Process_no Burst time Wait time Turn around time");
      for(i=0;i<n;i++)
      printf("\n%d\t %d\t %d\t %d",i+1,bt[i],wt[i],tat[i]);
     printf("\nAvg wait time is %f Avg turn around time is %f",awt,atat);
      getch();
      }
```

Input:

Enter no of jobs 4

Enter burst time 5 12 8 20

Output:

Bt wt tt 5 0 5

12 5 13

8 13 25

20 25 45 aw=10.75000 at=22.000000

VIVA QUESTIONS:

1.Round Robin scheduling is used in
(A)Disk scheduling. (B)CPU scheduling (C)I/O scheduling. (D)Multitasking

2. What are the dis-advantages of RR Scheduling Algoritm?

3. What are the advantages of RR Scheduling Algoritm?

4. Super computers typically employ_____.
1 Real time Operating system 2 Multiprocessors OS 3 desktop OS 4 None of the above

5.An optimal scheduling algorithm in terms of minimizing the average waiting time of a given set of processes is
.
1 FCFS scheduling algorithm 2 Round robin scheduling algorithm 3 Shortest job - first scheduling algorithm 4 None of the above

EXPERIMENT : 4

Objective: Simulate the following CPU Scheduling Algorithms

Priority Scheduling

THEORY:

In Priority Scheduling, each process is given a priority, and higher priority methods are executed first, while equal priorities are executed First Come First Served or Round Robin.

There are several ways that priorities can be assigned:

- Internal priorities are assigned by technical quantities such as memory usage, and file/IO operations.
- External priorities are assigned by politics, commerce, or user preference, such as importance and amount being paid for process access (the latter usually being for mainframes).

ALGORITHM

1. Start
2. Declare the array size
3. Read the number of processes to be inserted
4. Read the Priorities of processes
5. sort the priorities and Burst times in ascending order

   calculate the waiting time of each process wt[i+1]=bt[i]+wt[i]

   calculate the turnaround time of each process tt[i+1]=tt[i]+bt[i+1]
6. Calculate the average waiting time and average turnaround time.
7. Display the values
8. Stop

PROGRAM:

```
#include<stdio.h>
 #include<conio.h>
void main()
{
int i,j,pno[10],prior[10],bt[10],n,wt[10],tt[10],w1=0,t1=0,s;
float aw,at;
clrscr();
printf("enter the number of processes:");
scanf("%d",&n);
for(i=0;i<n;i++)
{
printf("The process %d:\n",i+1); printf("Enter the burst time of processes:");
scanf("%d",&bt[i]);
printf("Enter the priority of processes %d:",i+1); scanf("%d",&prior[i]);
pno[i]=i+1;
}
for(i=0;i<n;i++)
{
for(j=0;j<n;j++)
{
if(prior[i]<prior[j])
{
s=prior[i]; prior[i]=prior[j];
prior[j]=s;
```

```
 s=bt[i]; bt[i]=bt[j];
  bt[j]=s;
 s=pno[i]; pno[i]=pno[j]; pno[j]=s;
 }
 }
 }
 for(i=0;i<n;i++)
 {
 wt[0]=0;
 tt[0]=bt[0];
 wt[i+1]=bt[i]+wt[i];
 tt[i+1]=tt[i]+bt[i+1];  w1=w1+wt[i];
 t1=t1+tt[i];
 aw=w1/n;
 at=t1/n;
 }
printf(" \n job \t bt \t wt \t tat \t prior\n");
 for(i=0;i<n;i++)
 printf("%d \t %d \t %d\t %d\t %d\n",pno[i],bt[i],wt[i],tt[i],prior[i]);
 printf("aw=%f \t at=%f \n",aw,at);
 getch();
 }
```

Input:

Enter no of jobs 4

Enter bursttime 10 2 4 7

Enter priority values 4 2 1 3

Output:

Bt prioriywttt 4 1 0 4

2 2 4 6

7 3 6 13

10 4 13 23 aw=5.750000 at=12.500000

VIVA QUESTIONS:

1. Priority CPU scheduling would most likely be used in a_____os.
2. Cpu allocated process to_____priority.
3. calculate avg waiting time=
4. Maximum CPU utilization obtained with _____
5. Using_____algorithms find the min & max waiting time.

Experiment-5

Objective: Write a C program for Contiguous Memory Allocation On Worst Fit

Description:- In this technique, we check all the blocks in sequential order. So, the first process and compare its size with the first size of the block. If it is less we move to the other block and so on until all the processes are allocated.

First Fit Algorithm

1. Read the number of processes and number of the block from the user
2. Read the size of each block and the size of all the process requests.
3. Start allocating the processes
4. Display the results as shown below
5. Stop

Program

```
#include <stdio.h>
int main()
{
int a[10], b[10], a1, b1, flags[10], all[10];
 int i, j; printf("\n\t\t\tMemory Management" " Scheme -" " First Fit\n");
 for (i = 0; i < 10; i++)
 {
flags[i] = 0; all[i] = -1;
}
 printf("Enter number of blocks: ");
scanf("%d", &a1);
 printf("\nEnter the size of each" " block:\n "); for (i = 0; i < a1; i++)
{
printf("Block no.%d: ", i);
scanf("%d", &a[i]);
}
```

```
printf("\nEnter no. of " "processes: "); scanf("%d", &b1);


printf("\nEnter size of each" " process:\n ");
for (i = 0; i < b1; i++)
{
printf("Process no.%d: ", i);
 scanf("%d", &b[i]);
}
 for (i = 0; i < b1; i++) for (j = 0; j < a1; j++) if (flags[j] == 0 && a[j] >= b[i])
{
all[j] = i; flags[j] = 1;
break;
}
printf("\nBlock no.\tsize\t\t" "process no.\t\tsize");
for (i = 0; i < a1; i++)
 {
printf("\n%d\t\t%d\t\t", i + 1, a[i]);
if (flags[i] == 1)
{
printf("%d\t\t\t%d", all[i] + 1, b[all[i]]); } else printf("Not allocated");
}
printf("\n"); }
```

Output:

```
                    Memory Management Scheme - First Fit
Enter number of blocks: 3

Enter the size of each block:
 Block no.0: 12
Block no.1: 9
Block no.2: 7

Enter no. of processes: 3

Enter size of each process:
 Process no.0: 5
Process no.1: 4
Process no.2: 9

Block no.       size            process no.             size
1               12              1                       5
2               9               2                       4
3               7               Not allocated
```

Experiment-6

Objective:- Write a C program for Contiguous Memory Allocation On Best Fit

Theory

This memory management technique uses the best memory block based on the process memory request. So, here we select the smallest block and fulfill the memory request by the incoming process. Therefore memory used in this technique is optimally used. The disadvantage is that as it compares the block with all the memory sizes it increases the time requirement. So, this technique is slower as compared to the rest.

Algorithm

1. Read the number of processes and number of blocks from the user
2. Get the size of each block and process requests
3. Then select the best memory block
4. Display the result as shown below
5. The fragmentation column will keep track of wasted memory
6. Stop

Programming

```
#include <stdio.h>
int main()
{
int a[20], b[20], c[20], b1, c1;
int i, j, temp;
static int barr[20], carr[20];
printf("\n\t\t\tMemory Management"
" Scheme - Best Fit");
printf("\nEnter the number of "
"blocks:");
scanf("%d", &b1);
```

```
printf("Enter the number of"
" processes:");
scanf("%d", &c1);
int lowest = 9999;
printf("\nEnter the size of the"
" blocks:\n");
for (i = 1; i <= b1; i++)
{
printf("Block no.%d:", i);
scanf("%d", &b[i]);
}
printf("\nEnter the size of"
" the processes :\n");
for (i = 1; i <= c1; i++)
{
printf("Process no.%d:", i);
scanf("%d", &c[i]);
}
for (i = 1; i <= c1; i++)
{
for (j = 1; j <= b1; j++)
{
if (barr[j] != 1)
{
temp = b[j] - c[i];
if (temp >= 0)
if (lowest > temp)
{
carr[i] = j;
lowest = temp;
```

```
}
}
}
a[i] = lowest;
barr[carr[i]] = 1;
lowest = 10000;
}
printf("\nProcess_no\tProcess"
"_size\tBlock_no\t"
"Block_size\tFragment");
for (i = 1; i <= c1 && carr[i] != 0; i++)
{
printf("\n%d\t\t%d\t\t%d\t\t"
"%d\t\t%d", i,
c[i], carr[i], b[carr[i]], a[i]);
}
printf("\n");
}
```

Output

```
                    Memory Management Scheme - Best Fit
Enter the number of blocks:5
Enter the number of processes:4

Enter the size of the blocks:
Block no.1:9
Block no.2:13
Block no.3:5
Block no.4:8
Block no.5:2

Enter the size of the processes :
Process no.1:3
Process no.2:4
Process no.3:8
Process no.4:14

Process_no      Process_size    Block_no        Block_size      Fragment
1               3               3               5               2
2               4               4               8               4
3               8               1               9               1
```

Experiment-7

Objective: Write a C program for Contiguous Memory Allocation On First Fit algorithm?

Theory:- The First Fit memory allocation checks the empty memory blocks in a sequential manner. It means that the memory Block which found empty in the first attempt is checked for size.

Algorithm:

Step: START.

2 Step: At first get the no of processes and blocks.

3 Step: Allocate the process by **if(size of block>=size of the process)** then allocate the process else move to the next block.

4 Step: Now Display the processes with blocks and allocate to respective process.

5 Step: STOP.

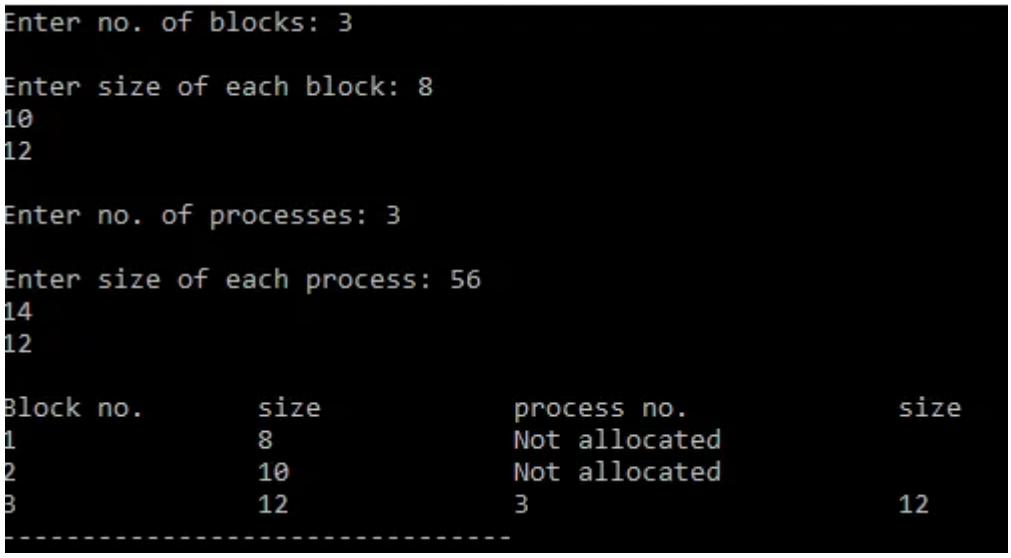
Program:

```
#include<stdio.h>

void main()

{

    int bsize[10], psize[10], bno, pno, flags[10], allocation[10], i, j;

    for(i = 0; i < 10; i++)

    {

        flags[i] = 0;

        allocation[i] = -1;

    }

    printf("Enter no. of blocks: ");

    scanf("%d", &bno);
```

```
printf("\nEnter size of each block: ");
for(i = 0; i < bno; i++)
        scanf("%d", &bsize[i]);
printf("\nEnter no. of processes: ");
scanf("%d", &pno);
printf("\nEnter size of each process: ");
for(i = 0; i < pno; i++)
        scanf("%d", &psize[i]);
for(i = 0; i < pno; i++)        //allocation as per first fit
        for(j = 0; j < bno; j++)
                if(flags[j] == 0 && bsize[j] >= psize[i])
                {
                        allocation[j] = i;
                        flags[j] = 1;
                        break;
                }
//display allocation details
printf("\nBlock no.\tsize\t\tprocess no.\t\tsize");
for(i = 0; i < bno; i++)
{
        printf("\n%d\t\t%d\t\t", i+1, bsize[i]);
        if(flags[i] == 1)
                printf("%d\t\t\t%d",allocation[i]+1,psize[allocation[i]]);
```

```
            else
                printf("Not allocated");
        }
}
```

Output:



```
Enter no. of blocks: 3

Enter size of each block: 8
10
12

Enter no. of processes: 3

Enter size of each process: 56
14
12

Block no.       size            process no.             size
1               8               Not allocated
2               10              Not allocated
3               12              3                       12
--------------------------------
```

Experiment-8

Objective:- Write a C program for FIFO Page Replacement Algorithm in C Program

Theory:-

Generally, most operating systems use the method of paging for memory management. These algorithms are needed to make a decision of which page needs to be replaced when a new page comes into the picture or is demanded.

The need for the demand occurs when the Operating Systems need any page for the processing which is not actually present in the main memory. This situation is also called Page Fault.

Page Fault: The page fault takes place when the main program accesses the memory page which is mapped into a virtual address space but is not loaded in physical memory.

When the Physical Memory is much smaller than the Virtual Memory in such a situation Page fault happens

FIFO which is also called First In First Out is one of the types of Replacement Algorithms. This algorithm is used in a situation where an Operating system replaces an existing page with the help of memory by bringing a new page from the secondary memory.

FIFO is the simplest among all algorithms which are responsible for maintaining all the pages in a queue for an operating system and also keeping track of all the pages in a queue.

The older pages are kept in the front and the newer ones are kept at the end of the queue. Pages that are in the front are removed first and the pages which are demanded are added.

Example of FIFO Replacement Algorithm

Now consider a page reference string 1, 3, 0, 3, 5, 6 with 3-page frames. So find the number of page faults?

initially, all the slots are empty so the first 3 elements 1, 3, 0 are allocated to the empty slots. So **Page Faults = 3**

After 1, 3, 0 element 3 comes. But 3 is already in memory So **Page Faults = 0**

Now element 5 comes. It is not present in memory so the oldest page slot 1 is replaced. So **Page Faults = 1**

After 5 element 6 comes, It is also not available in the memory so it gets replaced with the oldest page slot 3. So **Page Faults = 1**

At last element 3 comes which is not available in memory so it replaces element 0. So **page faults = 1**

FIFO Page Replacement Algorithm

1. Start traversing the pages.
2. Now declare the size w.r.t length of the Page.
3. Check need of the replacement from the page to memory.
4. Similarly, Check the need of the replacement from the old page to new page in memory.
5. Now form the queue to hold all pages.
6. Insert Require page memory into the queue.
7. Check bad replacemets and page faults.
8. Get no of processes to be inserted.
9. Show the values.
10. Stop

Program

```
#include <stdio.h>
int main()
{
int referenceString[10], pageFaults = 0, m, n, s, pages, frames;
printf("\nEnter the number of Pages:\t");
scanf("%d", &pages);
printf("\nEnter reference string values:\n");


for( m = 0; m < pages; m++)
{
  printf("Value No. [%d]:\t", m + 1);
  scanf("%d", &referenceString[m]);
}
printf("\n What are the total number of frames:\t");
{
  scanf("%d", &frames);
}
int temp[frames];
for(m = 0; m < frames; m++)
{
 temp[m] = -1;
}
for(m = 0; m < pages; m++)
{
 s = 0;
 for(n = 0; n < frames; n++)
  {
```

```
        if(referenceString[m] == temp[n])
          {
            s++;
            pageFaults--;
          }
      }
      pageFaults++;
      if((pageFaults <= frames) && (s == 0))
        {
          temp[m] = referenceString[m];
        }
      else if(s == 0)
        {
          temp[(pageFaults - 1) % frames] = referenceString[m];
        }
        printf("\n");
        for(n = 0; n < frames; n++)
         {
           printf("%d\t", temp[n]);
         }
    }
    printf("\nTotal Page Faults:\t%d\n", pageFaults);
    return 0;
    }
```

Output:

Experiment:9

Objective: To write a program in C to Implement LRU Page Replacement Algorithm.

Theory:

- Page replacement algorithms are used to decide what pages to page out when a page needs to be allocated.
- This happens when a page fault occurs and a free page cannot be used to satisfy the allocation

Types:

- FIFO replacement
- LRU replacement
- Optimal replacement

LRU:

- "Replace *the page that had not been used* for a longer sequence of time".
- The frames are empty in the beginning and initially no page fault occurs so it is set to zero. When a page fault occurs the page reference sting is brought into the memory.
- The operating system keeps track of all pages in the memory, thereby keeping track of the page that had not been used for longer sequence of time.
- If the page in the page reference string is not in memory, the page fault is incremented and the page that had not been used for a longer sequence of time is replaced. If the page in the page reference string is in the memory take the next page without calculating the next page.
- Take the next page in the page reference string and check if the page is already present in the memory or not.
- Repeat the process until all pages are referred and calculate the page fault for all those pages in the page references string for the number of available frames.

Algorithm:

- 1- Start traversing the pages.

1. i) If set holds less pages than capacity.

- a) Insert page into the set one by one until the size of set reaches capacity or all page requests are processed.
- b) Simultaneously maintain the recent occurred index of each page in a map called indexes.
- c) Increment page fault

2. ii) Else If current page is present in set, do nothing.

- Else

A. a) Find the page in the set that was least recently used. We find it using index array. We basically need to replace the page with minimum index.

B. b) Replace the found page with current page.

C. c) Increment page faults.

D. d) Update index of current page.

- 2 – Return page faults.

Program:

```
#include <stdio.h>
#include<stdbool.h>
int rs[100],count =0;
struct max{
int f[10];
bool lu[10];
}s;
struct min{
int buf[10];
int mat[10];
}nat;
bool dataavi(int n)
{
int i;
for(i=0;i<n;i++)
{
if(rs[count]==s.f[i])
return 1;
}

return 0;
}
int check(int n,int m)
{
int i,j=0;
for(j=0;j<n;j++)
{
int x=0;
```

```
for(i=count+1;i<m;i++)
{

if(s.f[j]==rs[i])
{
nat.buf[j]=i;
nat.mat[j]=j;
i=m+1;
x=1;
}
}
if(x==0&&s.lu[j]==0)
return j;
}

for (i = 0; i < n-1; i++)
{
for (j = 0; j < n-i-1; j++)
{
if (nat.buf[j] < nat.buf[j+1])
{
int temp=nat.buf[j];
nat.buf[j]=nat.buf[j+1];
nat.buf[j+1]=temp;
temp=nat.mat[j];
nat.mat[j]=nat.mat[j+1];
nat.mat[j+1]=temp;
}
```

```
}}
for(i=0;i<n;i++)
{
if(s.lu[nat.mat[i]]==0)
{
return nat.mat[i];
}
}
}

void lru (int n,int m)
{
int fs=0,i=0,in=0,kom=0;
for(i=0;i<n;i++)
s.lu[i]=false;
s.lu[n-1]=true;
while(count<m)
{
if(in<n)
{
if(dataavi(n)&&in>0)
{
count++;kom=0;
}
else
{
s.f[in++]=rs[count]; kom=1;
fs++;
count++;
```

```
}
}
else
{
if(dataavi(n))
{
count++; kom=0;
}
else
{
int j=check(n,m);
s.f[j]=rs[count++];
fs++; kom=1;
for(i=0;i<n;i++)
s.lu[i]=false;
s.lu[j]=true;
}
}
if(kom==1)
printf(" Page Fault :");
else
printf("            :");
for(i=0;i<n;i++)
printf(" %d",s.f[i]);
printf("\n");

}
printf("\n\npage faults =%d",fs);
}
```

```
int main()
{
int n ,m,i;

printf("ENTER THE NO OF FRAME AND REFERENCE STREAM\n");
scanf ("%d%d",&n,&m);
printf ("ENTER THE REFERENCE STREAM \n");
for(i=0;i<m;i++)
scanf("%d",&rs[i]);
lru (n,m);
return 0;
}
```

Execution:

```
Input:
3 20 1 2 3 4 2 1 5 6 2 1 2 3 7 6 3 2 1 2 3 6


Output:

ENTER THE NO OF FRAME AND REFERENCE STREAM
ENTER THE REFERENCE STREAM
 Page Fault : 1 0 0
 Page Fault : 1 2 0
 Page Fault : 1 2 3
 Page Fault : 4 2 3
          : 4 2 3
 Page Fault : 4 2 1
 Page Fault : 5 2 1
 Page Fault : 5 2 6
          : 5 2 6
```

```
Page Fault : 1 2 6
        : 1 2 6
Page Fault : 1 3 6
Page Fault : 7 3 6
        : 7 3 6
        : 7 3 6
Page Fault : 7 3 2
Page Fault : 1 3 2
        : 1 3 2
        : 1 3 2
Page Fault : 1 6 2


page faults =13
```

Experiment 10

Objective:-C Program to simulate Optimal Page Replacement C program

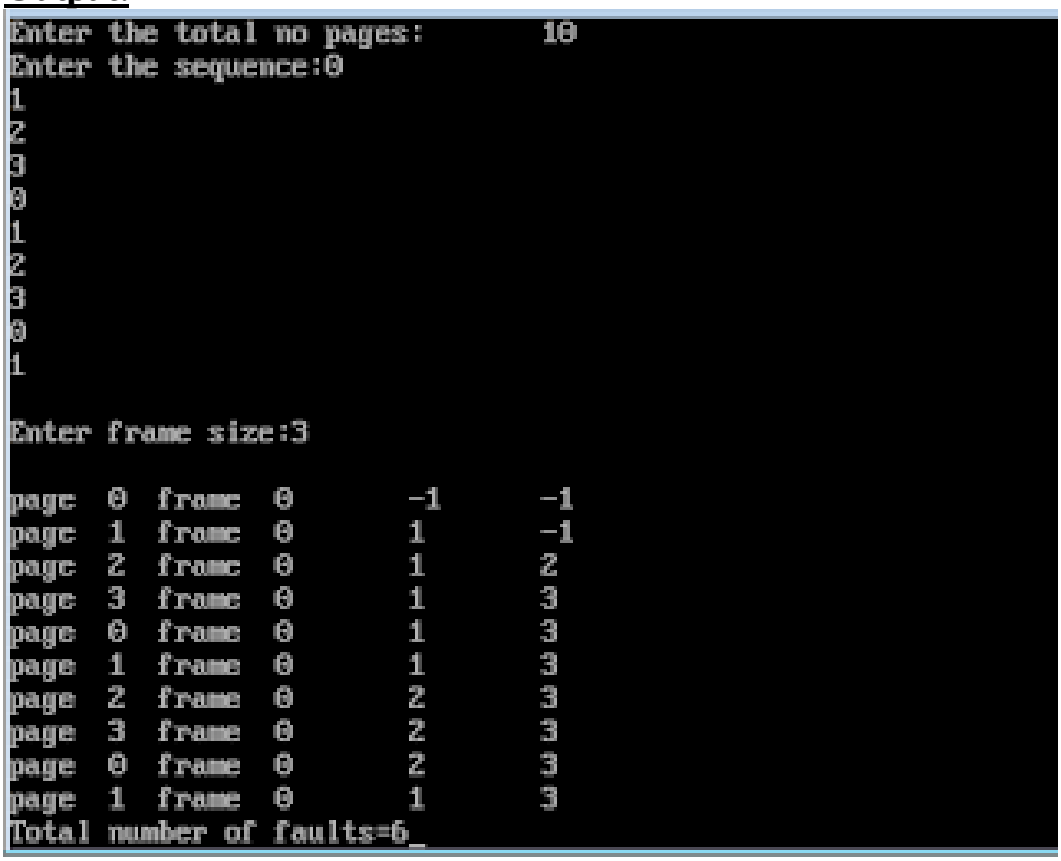
Theory: Optimal Page Replacement refers to the removal of the page that will not be used in the future, for the longest period of time.

Program Code:

```
#include<stdio.h>
int main()
{
int n,pg[30],fr[10];
int count[10],i,j,k,fault,f,flag,temp,current,c,dist,max,m,cnt,p,x;
fault=0;
dist=0;
k=0;
printf("Enter the total no pages:\t");
scanf("%d",&n);
printf("Enter the sequence:");
for(i=0;i<n;i++)
scanf("%d",&pg[i]);
printf("\nEnter frame size:");
scanf("%d",&f);

for(i=0;i<f;i++)
{
count[i]=0;
fr[i]=-1;
}
for(i=0;i<n;i++)
{
flag=0;
temp=pg[i];
for(j=0;j<f;j++)
{
if(temp==fr[j])
{
flag=1;
break;
}
}
```

```
if((flag==0)&&(k<f))
{
fault++;
fr[k]=temp;
k++;
}
else if((flag==0)&&(k==f))
{
fault++;
for(cnt=0;cnt<f;cnt++)
{
current=fr[cnt];
for(c=i;c<n;c++)
{
if(current!=pg[c])
count[cnt]++;
else
break;
}
}
max=0;
for(m=0;m<f;m++)
{
if(count[m]>max)
{
max=count[m];
p=m;
}
}
fr[p]=temp;
}
printf("\npage %d frame\t",pg[i]);
for(x=0;x<f;x++)
{
printf("%d\t",fr[x]);
}
}
printf("\nTotal number of faults=%d",fault);
return 0;
}
```

Output:


Enter the total no pages: 10
Enter the sequence:0
1
2
3
0
1
2
3
0
1
Enter frame size:3
page 0 frame 0 -1 -1
page 1 frame 0 1 -1
page 2 frame 0 1 2
page 3 frame 0 1 3
page 0 frame 0 1 3
page 1 frame 0 1 3
page 2 frame 0 2 3
page 3 frame 0 2 3
page 0 frame 0 2 3
page 1 frame 0 1 3
Total number of faults=6