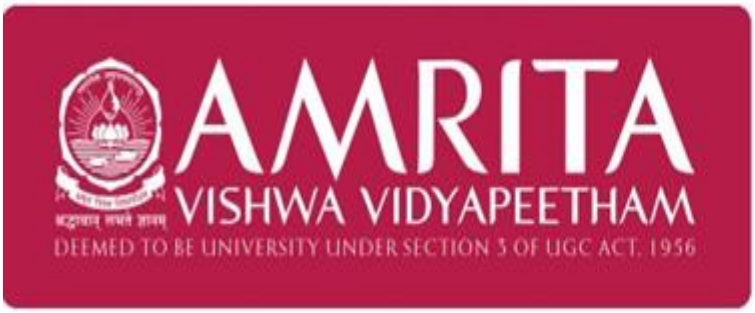


SCHOOL OF COMPUTING CHENNAI

LAB MANUAL

20CYS281 – OPERATING SYSTEM

B. TECH (II YEAR)

CYS (2023-2027)

TABLE OF CONTENTS

9.	Page Replacement Algorithms i. To write a program to execute First in First out page replacement Algorithm. ii. To write a program to execute Least recently Used Page Replacement Algorithm. iii. To write a program to execute Optimal Page Replacement Algorithm.
10.	Paging Memory Management To write a program to simulate paging technique of memory management.

Linux Commands:

- mkdir – To create a new directory
- cd – To move to a already exsisting directory
- cd.. – To go to parent directory
- ls – List the items present in the directory
- ls -l – List the items along with their priveleges
- ls -s – List the items along with their size
- cat > filename – To create new file
- cat filename – To view the already exsisting file
- cp source_file destination_file – To copy a file
- mv source_file destination_directory – To move a file from one directoryto another
- chmod 777 filename – To change the priveleges of a file
- date – to show date
- rm filename – To remove a file
- rmdir directoryname – To remove a directory
- ifconfig – To know details about the machine
- ping – To check the connectivity with a machine
- pwd – To view the present working directory
- ps – To view the process ID and time taken for a process
- time – shows a time
- top – To view the ongoing processes

Shell Scripting:

01. Write a shell script program to print your name, dept, reg no.

```
┌──(kali㉿kali)-[~/Documents/shell]
└─$ ./1.sh
Name:Saravanan
Department:CYS
Roll.no:CH.EN.U4CYS21074
```

02. Write a. shell script program to print using all arithmetic operators

```
┌──(kali㉿kali)-[~/Documents/shell]
└─$ cat 2.sh
echo "Enter a number:"
read a
echo "Enter another number:"
read b
echo "Addition:$a+$b=$(($a+$b))"
echo "Subraction:$a-$b=$(($a-$b))"
echo "Multiplication:$a x $b=$(($a*$b))"
echo "Division:$a/$b=$(($a/$b))"
echo "Mod:$a%$b=$(($a%$b))"

┌──(kali㉿kali)-[~/Documents/shell]
└─$ ./2.sh
Enter a number:
3
Enter another number:
3
Addition:3+3=6
Subraction:3-3=0
Multiplication:3 x 3=9
Division:3/3=1
Mod:3%3=0
```

03. Write a script program using While loop.

```
┌──(kali㉿kali)-[~/Documents/shell]
└─$ cat 3.sh
num=5
echo "Number is $num"
while [ $num -le 10 ]
do
        echo "$num"
        num=$(($num+1))
done

┌──(kali㉿kali)-[~/Documents/shell]
└─$ ./3.sh
Number is 5
5
6
7
8
9
10
```

04. Write a script for using For loop to print sum of n natural numbers

```
┌──(kali㉿kali)-[~/Documents/shell]
└─$ cat 4.sh
#!/bin/bash
echo "Enter Size"
read N
sum=0
echo "Enter Numbers"
for (( i=1; i<=N; i++ ))
do
  read num
  sum=$(( sum + num ))
done
echo "Sum of the natural number is $sum"

┌──(kali㉿kali)-[~/Documents/shell]
└─$ ./4.sh
Enter Size
2
Enter Numbers
3
3
Sum of the natural number is 6
```

05. i.) Write a shell script to check given no is one digit number or two-digit number using if condition.

```
┌──(kali㉿kali)-[~/Documents/shell]
└─$ cat 5.sh
echo "Enter either one digit or two digit  number"
read num
if [ $num -lt 10 ]
then
        echo "The given number $num is one digit"
else
        echo "The given number $num is two digit"
fi


┌──(kali㉿kali)-[~/Documents/shell]
└─$ ./5.sh
Enter either one digit or two digit  number
65
The given number 65 is two digit
```

ii.) Using switch case create a new file, delete a file and see the content of the file.

```
┌──(kali㉿kali)-[~/Documents/shell]
└─$ cat 5b.sh
#!/bin/bash
echo "1.Create New File
2.Read File
3.Delete File"
echo "ENter your choise"
read choise
case $choise in
        1)
                echo "Enter New File Name:"
                read file
                echo "Hello" > $file
                echo "New File Created" > $file
                echo "File Created";;
        2)
                echo "Enter File Name:"
                read file
                cat $file;;
        3)
                echo "Enter File Name to Remove:"
                read fn
                rm -i $fn;;

        *)
                echo "Incorrect Choise";;
esac


┌──(kali㉿kali)-[~/Documents/shell]
└─$ bash 5b.sh
1.Create New File
2.Read File
3.Delete File
ENter your choise
1
Enter New File Name:
file1.txt
File Created
```

06. Using If condition with AND operator to validate the username and password

```
┌──(kali㉿kali)-[~/Documents/shell]
└─$ cat 6.sh
#! /bin /bash
user="adhi"
pass="1234"
echo "Enter Username"
read u
echo "Enter Password"
read p
if [[ ( $u == $user && $p == $pass ) ]]
then
        echo "Valid"
else
        echo "Invalid"
fi


┌──(kali㉿kali)-[~/Documents/shell]
└─$ bash 6.sh
Enter Username
adhi
Enter Password
1234
Valid

┌──(kali㉿kali)-[~/Documents/shell]
└─$ bash 6.sh
Enter Username
adhi
Enter Password
1223
Invalid
```

07. Create a function to print factorial of n numbers.

```
┌──(kali㉿kali)-[~/Documents/shell]
└─$ cat 7.sh
#!/bin/bash
fact()
{
        product=1
        while [ $num -gt 1 ]
        do
                product=$((product * num))
                num=$((num - 1))
        done
        echo "Factorial :"
        echo $product
}

echo "Enter the number:"
read num
if  [ num == 0 ]
then
        echo 1
else
        fact $num
fi

┌──(kali㉿kali)-[~/Documents/shell]
└─$ bash 7.sh
Enter the number:
4
Factorial :
24
```

08. Write a program to make new directory and check the directory is already exist or not.

```
┌──(kali㉿kali)-[~/Documents/shell]
└─$ cat 8.sh
#!/bin/bash
echo -e "Enter the name of the directory : \c"
read dir_name
if [ -e $dir_name ]
then
        echo "$dir_name exist"
else
        mkdir $dir_name
        echo "$dir_name created"
fi

┌──(kali㉿kali)-[~/Documents/shell]
└─$ bash 8.sh
Enter the name of the directory : samp
samp created

┌──(kali㉿kali)-[~/Documents/shell]
└─$ ls
1.sh  2.sh  3.sh  4]  4.sh  5]  5b.sh  5.sh  6.sh  7.sh  8.sh  num1]  num2]  samp  sample.txt
```

09. Write a shell script that list all the files in the directory.

```
┌──(kali㉿kali)-[~/Documents]
└─$ cat 9.sh
#!/bin/bash
echo " Enter directory name : "
read dir
for file in "$dir/"*
do
        [[ -f "$file" ]] && echo "$file"
done


┌──(kali㉿kali)-[~/Documents]
└─$ bash 9.sh
 Enter directory name :
shell
shell/1.sh
shell/2.sh
shell/3.sh
shell/4]
shell/4.sh
shell/5]
shell/5b.sh
shell/5.sh
shell/6.sh
shell/7.sh
shell/8.sh
shell/num1]
shell/num2]
shell/sample.txt
```

10. Write a shell script display list of all the files in the current directory to which the user has read, write and execute permissions.

-f $file	→	returns true if the file exists
-r $file	→	returns true if the file has read permission
-w $file	→	if the file has write permission
-x $file	→	if the file has execute permission
-a	→	checking multiple conditions same as && operator

```
┌──(kali㉿kali)-[~/Documents/shell]
└─$ cat 10.sh
#!/bin/bash

for file in *
do
        if [ -f $file ]
        then
                if [ -r $file -a -w $file -a -x $file ]
                then
                        ls -l $file
                fi
        fi
done

┌──(kali㉿kali)-[~/Documents/shell]
└─$ bash 10.sh
-rwxrwxrwx 1 kali kali 131 Nov  1 09:27 10.sh
-rwxrwxrwx 1 kali kali 113 Oct 31 04:19 1.sh
-rwxrwxrwx 1 kali kali 236 Oct 31 04:34 2.sh
-rwxrwxrwx 1 kali kali 88 Oct 31 04:52 3.sh
-rwxrwxrwx 1 kali kali 184 Oct 31 05:23 4.sh
-rwxrwxrwx 1 kali kali 402 Nov  1 09:02 5b.sh
-rwxrwxrwx 1 kali kali 179 Oct 31 05:33 5.sh
-rwxrwxrwx 1 kali kali 180 Oct 31 05:47 6.sh
-rwxrwxrwx 1 kali kali 233 Nov  1 09:11 7.sh
-rwxrwxrwx 1 kali kali 173 Nov  1 09:17 8.sh
```

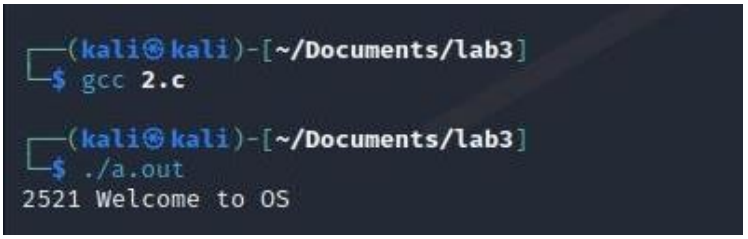
System Calls:

01. To write a program to get the Process ID in UNIX.

CODE:

```
#include <stdio.h>
#include <unistd.h>
#include <sys/stat.h>
int main()
{
    printf("%d Welcome to the OS Lab",getpid());
    return 0;
}
```

Output:

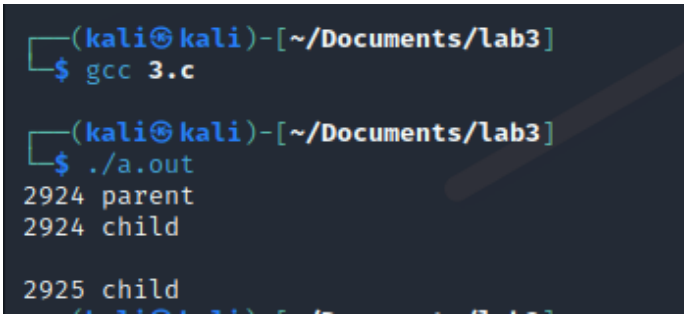


02. To write a program to execute Fork() system call and get the Process ID in UNIX.

CODE:

```
#include<stdio.h>
#include <unistd.h>
#include<sys/stat.h>
int main()
{
    printf("%d Parent PID",getpid());
    fork();
    printf("%d Child PID",getpid());
    return 0;
}
```

Output:

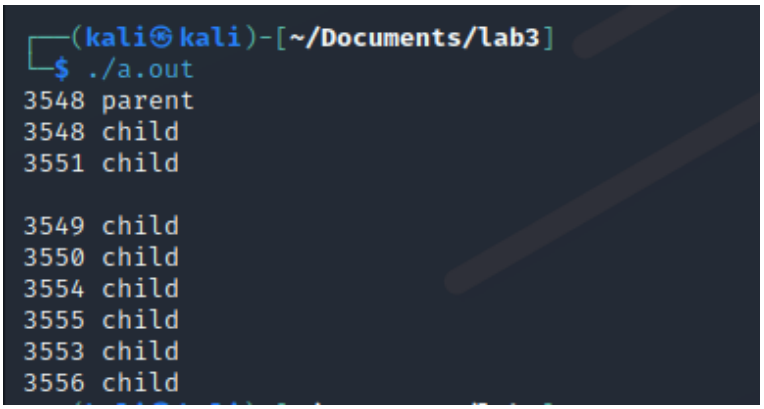


03. To write a program to calculate number of times a message is printed using fork() system call and process ID.

CODE:

```
#include<stdio.h>
#include <unistd.h>
#include<sys/stat.h>
int main()
{
    printf("%d Parent PID",getpid());
    fork();
    printf("%d Child PID",getpid());
    return 0;
}
```

Output:



04. To write a program to calculate number of times a message is printed using fork() system and apply OR command.

CODE:

```
#include<stdio.h>
#include<unistd.h>
#include<sys/stat.h>
int main()
{
    if(fork() || fork())
        printf("%d True.\n",getpid());
    else
        printf("%d False.\n",getpid());
    return 0;
}
```

Output:

```
┌──(kali㉿kali)-[~/Documents/lab3]
└─$ gcc 6.c

┌──(kali㉿kali)-[~/Documents/lab3]
└─$ ./a.out
4838 True

4839 True
4841 False
```

05. To write a program to calculate number of times a message is printed using fork() system and apply AND command.

CODE:

```
#include<stdio.h>
#include<unistd.h>
#include<sys/stat.h>
int main()
{
    if(fork() && fork())
        printf("%d True.\n",getpid());
    else
        printf("%d False.\n",getpid());
    return 0;
}
```

Output:

```
┌──(kali㉿kali)-[~/Documents/lab3]
└─$ gcc 5.c

┌──(kali㉿kali)-[~/Documents/lab3]
└─$ ./a.out
5734 True
5735 False
5736 False
```

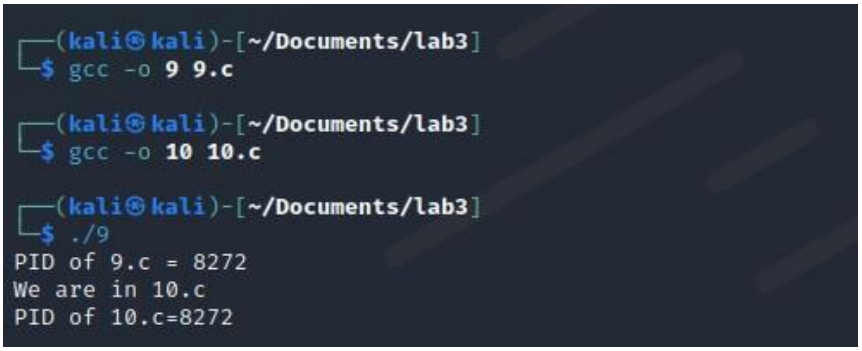
06. Write a Program using exec() system call.

CODE:

```c
// 9.c
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
int main(int argc, char *argv[])
{
    printf("PID of 9.c = %d\n", getpid());
    char *args[] = {"Hello","C","Programming",NULL};
    execv("./10",args);
    printf("Back to 9.c");
    return 0;
}
```

```c
// 10.c
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
int main(int argc, char *argv[])
{
    printf("We are in 10.c\n");
    printf("PID of 10.c = %d\n", getpid());
    return 0;
}
```

Output:



07. Write a program using wait() system call.

CODE:

```c
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
int main()
{
    if(fork() == 0)
        printf("hello from child\n");
    else
    {
        printf("hello from parent\n");
        wait(NULL);
        printf("child has terminated\n");
    }
    printf("Hi\n");
    return 0;
}
```

Output:

```
┌──(kali㉿kali)-[~/Documents/lab3]
└─$ ./a.out
hello from parent
hello from child
Hi
child has terminated
Hi
```

08. Write a C program using sleep() system call.

CODE:

```c
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
int main()
{
    fork();
    if(fork() == 0)
    {
        sleep(5);
        printf("This will finish after the parent\n");
    }
    else
        printf("This will finish below the child\n");
    return 0;
}
```

Output:

```
┌──(kali㉿kali)-[~/Documents/lab3]
└─$ gcc 8.c

┌──(kali㉿kali)-[~/Documents/lab3]
└─$ ./a.out
This will finish before the child

This will finish before the child
┌──(kali㉿kali)-[~/Documents/lab3]
└─$ This will finish after the parent
This will finish after the parent
```

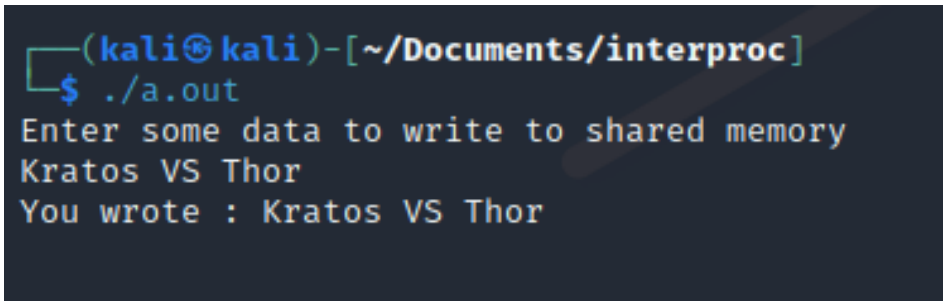
INTER – PROCESS COMMUNICATION:

01. To write a program to execute Inter Process Communication using Shared Memory.

CODE:

```c
#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|IPC_CREAT);
    shared_memory=shmat(shmid,NULL,0);
    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);
}
```

Output:



02. To write a program to execute Inter-Process Communication using Message Queue.

CODE:

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

```c
#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;
        }
    }
}
```

Output:

```
┌──(kali㉿kali)-[~/Documents/interproc]
└─$ ./a.out
Enter some text:
Cyber Security OS
Enter some text:
3rd Sem
Enter some text:
Engineering
Enter some text:
Amrita
Enter some text:
end
```

SCHEDULING ALGORITHM:

01. Round Robin Scheduling

CODE:

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


#define MAX 100
int queue[MAX];
int front = -1;
int rear = -1;

struct process {
    char name[4];
    int id; // Process id
    int at; // Arrival time
    int bt; // Burst time
    int rt; // Remaining time
    int ct; // Completion time
    int ta; // Turn Around time
    int wt; // Waiting time
```

```
    struct process* next;
} *P = NULL;


struct Gantt
{
    int id;
    int start;
    int stop;
    struct Gantt *next;
}*head = NULL;

int dequeue()
{
    int data;
    if(front == -1 || front == rear + 1)
    {
        printf("Queue underflow\n");
        exit(1);
    }
    data = queue[front];
    front++;
    return data;
}

void enqueue(int data)
{
    if(rear == MAX-1)
    {
        printf("Queue overflow\n");
        exit(1);
    }
    if(front == -1)
    front = 0;
    rear++;
    queue[rear] = data;
}

void update(int stop,int id)
{
    struct process *ptr = P;
    while(ptr != NULL)
```

```
    {
        if(ptr->id == id)
        {
            ptr->ct = stop;
        }
        ptr = ptr->next;
    }
}

void addGantt(int id, int start, int stop)
{
    struct Gantt *newNode = (struct Gantt*)malloc(sizeof(struct
Gantt));
    struct Gantt *ptr = head;
    newNode->id = id;
    newNode->start = start;
    newNode->stop = stop;
    newNode->next = NULL;
    if(head == NULL)
    head = newNode;
    else
    {
        while(ptr->next != NULL)
        {
            ptr=ptr->next;
        }
        ptr->next = newNode;
    }
    update(stop,id);
}

struct process* swap(struct process* ptr1, struct process* ptr2) {
    if (ptr1 && ptr2) {
        struct process* tmp = ptr2->next;
        ptr2->next = ptr1;
        ptr1->next = tmp;
        return ptr2;
    }
    return NULL;
}

void bubbleSort(int count) {
```

```
    struct process** head = &P;
    struct process** h;
    int i, j, swapped;

    for (i = 0; i < count - 1; i++) {

        h = head;
        swapped = 0;

        for (j = 0; j < count - i - 1; j++) {

            struct process* p1 = *h;
            struct process* p2 = p1->next;

            if (p1->at > p2->at) {

                /* update the link after swapping */
                *h = swap(p1, p2);
                swapped = 1;
            }

            h = &(*h)->next;
        }

        /* break if the loop ended without any swap */
        if (swapped == 0)
            break;
    }
}

void addProcess(char* name, int at, int bt,int id) {
    struct process* newNode = (struct process*)malloc(sizeof(struct
process));
    strcpy(newNode->name, name);
    newNode->at = at;
    newNode->bt = bt;
    newNode->rt = bt;
    newNode->id = id;
    newNode->ct = 0;
    newNode->ta = 0;
    newNode->wt = 0;
    newNode->next = NULL;
```

```
    if (P == NULL)
        P = newNode;
    else {
        struct process* temp = P;
        while (temp->next != NULL) {
            temp = temp->next;
        }
        temp->next = newNode;
    }
}

void printGantt()
{
    struct Gantt *temp = head;
    printf("%d",temp->id);
    temp = temp->next;
    while(temp != NULL)
    {
        printf(" -> %d",temp->id);
        temp = temp->next;
    }
}

void main()
{
    int n, i, tt = 0, quantum, at, bt;
    char name[4];
    printf("Enter the no. of processes : ");
    scanf("%d", &n);
    int AT[n], BT[n], RT[n], id[n];
    for (i = 0; i < n; i++)
    {
        printf("Enter the name of the process : ");
        scanf("%s", name);
        printf("Enter the Arrival time : ");
        scanf("%d", &at);
        printf("Enter the Burst time : ");
        scanf("%d", &bt);
        tt += bt;
        addProcess(name, at, bt,i);
        AT[i] = at;
        BT[i] = bt;
```

```
        RT[i] = bt;
        id[i] = i;
    }

    printf("Enter the time quantum : ");
    scanf("%d", &quantum);

    bubbleSort(n);

    i = 0;
    int temp_id,start = AT[0],stop = 0;
    enqueue(id[0]);
    while(i<tt)
    {
        temp_id = dequeue();
        if(RT[temp_id] >= quantum)
        {
            stop = stop + quantum;
            i = i + quantum;
            RT[temp_id] = RT[temp_id] - quantum;
        }
        else if(RT[temp_id] < quantum)
        {
            stop = stop + RT[temp_id];
            i = i + RT[temp_id];
            RT[temp_id] = 0;
        }
        addGantt(temp_id,start,stop);
        int j,k;
        for(j=start+1;j<=stop;j++)
        {
            for(k=1;k<n;k++)
            {
                if(AT[k] == j)
                {
                    enqueue(id[k]);
                }
            }
        }
        start = stop;
        if(RT[temp_id] != 0)
        enqueue(id[temp_id]);
```

```
    }

    double sumwt = 0.0, sumta = 0.0, sumct = 0.0;

    printGantt();

    struct process *ptr = P;
    printf("\nProcess name\tCT\tTA\tWT\n");
    while(ptr != NULL)
    {
        ptr->ta = ptr->ct - ptr->at;
        ptr->wt = ptr->ta - ptr->bt;
        sumct += ptr->ct;
        sumta += ptr->ta;
        sumwt += ptr->wt;
        printf("%s          \t%d \t%d \t%d\n",ptr->name,ptr->ct,ptr-
>ta,ptr->wt);
        ptr = ptr->next;
    }

    printf("\nAverage Completion time: %.2f\n", (double)sumct / n);
    printf("Average Turn Around Time: %.2f\n", (double)sumta / n);
    printf("Average Waiting time: %.2f\n", (double)sumwt / n);
}
```

Output:

```
┌──(kali㉿kali)-[~/Documents/algorithm]
└─$ ./a.out
 Total number of process in the system: 3

 Enter the Arrival and Burst time of the Process[1]
 Arrival time is:       0

Burst time is:  8

 Enter the Arrival and Burst time of the Process[2]
 Arrival time is:       0

Burst time is:  10

 Enter the Arrival and Burst time of the Process[3]
 Arrival time is:       0

Burst time is:  6
Enter the Time Quantum for the process:         2

 Process No              Burst Time              TAT             Waiting Time
Process No[3]            6                               18                      12
Process No[1]            8                               20                      12
Process No[2]            10                              24                      14
 Average Turn Around Time:      12.666667
 Average Waiting Time:  20.666666
```

02. Shortest Job First Scheduling.

PREEMPTIVE

CODE:

```
#include<stdio.h>
struct process
{
    int WT,AT,BT,TAT;
};
struct process a[10];

int main()
{
    int n,temp[10];
    int count=0,t=0,short_P;
    float total_WT=0, total_TAT=0,Avg_WT,Avg_TAT;
    printf("Enter the number of the process\n");
    scanf("%d",&n);
    printf("Enter the arrival time and burst time of the
process\n");
    for(int i=0;i<n;i++)
```

```
    {
        printf("Enter the arrival time of process[%d]",(i+1));
        scanf("%d",&a[i].AT);
        printf("Enter the burst time of process[%d]",i+1);
        scanf("%d",&a[i].BT);
        temp[i]=a[i].BT;
    }
    a[9].BT=10000;
    for(t=0;count!=n;t++)
    {
        short_P=9;
        for(int i=0;i<n;i++)
        {
            if(a[i].BT<a[short_P].BT && (a[i].AT<=t && a[i].BT>0))
            {
                short_P=i;
            }
        }
        a[short_P].BT=a[short_P].BT-1;

        // if any process is completed if(a[short_P].BT==0)
        {
            // one process complete count++;
            a[short_P].WT=t+1-a[short_P].AT-temp[short_P];
a[short_P].TAT=t+1-a[short_P].AT;

            // total calculation total_WT=total_WT+a[short_P].WT;
total_TAT=total_TAT+a[short_P].TAT;
        }
    }
    Avg_WT = total_WT/n;

    Avg_TAT = total_TAT/n;

    // printing of the answer
    printf("\nProcess Waiting Time Turn Around Time \n");
    for(int i=0;i<n;i++)
    {
        printf("%d\t\t%d\t\t%d\n",i+1,a[i].WT,a[i].TAT);
    }
    printf("Avg waiting time of the process is %f\n",Avg_WT);
    printf("Avg turn around time of the process %f\n",Avg_TAT);
```

```
}
```

Output:

```
┌──(kali㉿kali)-[~/Documents/algorithm]
└─$ ./a.out
Enter the number of the process
3
Enter the arrival time and burst time of the process
Enter the arrival time of process[1]0
Enter the burst time of process[1]16
Enter the arrival time of process[2]2
Enter the burst time of process[2]18
Enter the arrival time of process[3]5
Enter the burst time of process[3]10

Process Waiting Time Turn Around Time
1               10              26
2               24              42
3               0               10
Avg waiting time of the process is 11.333333
Avg turn around time of the process 26.000000
```

Non – Preemptive

CODE:

```
#include<stdio.h>
int main()
{
    int bt[20],p[20],wt[20],tat[20],i,j,n,total=0,pos,temp;
    float avg_wt,avg_tat;
    printf("Enter number of process:");

    scanf("%d",&n); printf("\nEnter Burst Time:\n");
    for(i=0;i<n;i++)
    {
        printf("p%d:",i+1);
        scanf("%d",&bt[i]); p[i]=i+1;
    }
    for(i=0;i<n;i++)
    {
        pos=i;
        for(j=i+1;j<n;j++)
        {
            if(bt[j]<bt[pos])
```

```
                pos=j;
            }
            temp=bt[i]; bt[i]=bt[pos]; bt[pos]=temp; temp=p[i];
p[i]=p[pos]; p[pos]=temp;
        }
        wt[0]=0;
        for(i=1;i<n;i++)
        {
            wt[i]=0;
            for(j=0;j<i;j++)
                wt[i]+=bt[j];
            total+=wt[i];
        }
        avg_wt=(float)total/n; total=0;

        printf("\nProcess\t Burst Time  \tWaiting Time\tTurnaround
Time");
        for(i=0;i<n;i++)
        {
            tat[i]=bt[i]+wt[i]; total+=tat[i];
            printf("\np%d\t\t
%d\t\t    %d\t\t\t%d",p[i],bt[i],wt[i],tat[i]);
        }
        avg_tat=(float)total/n;
        printf("\n\nAverage Waiting Time=%f",avg_wt);
        printf("\nAverage Turnaround Time=%f\n",avg_tat);
}
```

Output:

```
┌──(kali㉿kali)-[~/Documents/algorithm]
└─$ ./a.out
Enter number of process:3

Enter Burst Time:
p1:16
p2:18
p3:10

Process     Burst Time          Waiting Time    Turnaround Time
p3                10                  0                  10
p1                16                  10                 26
p2                18                  26                 44

Average Waiting Time=12.000000
Average Turnaround Time=26.666666
```

03. First Come First Serve Scheduling.

AIM:

To write a c program to simulate the CPU scheduling algorithm First Come FirstServe (FCFS)

DESCRIPTION:

To calculate the average waiting time using the FCFS algorithm first the waiting time of the first process is kept zero and the waiting time of the second process is the burst time of the first process and the waiting time of the third process is the sum of the burst times of the first and the second process and so on. After calculating all the waiting times the average waiting time is calculated as the average of all the waiting times. FCFSmainly says first come first serve the algorithm which came first will be served first.

ALGORITHM:

Step 1: Start the process
Step 2: Accept the number of processes in the ready Queue
Step 3: For each process in the ready Q, assign the process name and the burst time
Step 4: Set the waiting of the first process as _0'and its burst time as its turnaround time
Step 5: for each process in the Ready Q calculate

a. Waiting time (n) = waiting time (n-1) + Burst time (n-1) b). Turnaround time (n)= waiting time(n)+Burst time(n)

Step 6: Calculate

a)Average waiting time = Total waiting Time / Number of process
b)Average Turnaround time = Total Turnaround Time / Number of

processStep 7: Stop the process

CODE:

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

```c
int main()
{
    int n,bt[20],wt[20],tat[20],avwt=0,avtat=0,i,j;
    printf("Enter total number of processes(maximum 20):");
    scanf("%d",&n);
    printf("Enter Process Burst Time\n");
    for(i=0;i<n;i++)
    {
        printf("P[%d]:",i+1);
        scanf("%d",&bt[i]);
    }
    wt[0]=0;
    for(i=1;i<n;i++)
    {
        wt[i]=0;
        for(j=0;j<i;j++)
            wt[i]+=bt[j];
    }
    printf("\nProces\tBurst Time\tWaiting Time\tTurnaround Time");
    for(i=0;i<n;i++)
    {
        tat[i]=bt[i]+wt[i]; avwt+=wt[i]; avtat+=tat[i];
        printf("\nP[%d]\t\t%d\t\t%d\t\t%d",i+1,bt[i],wt[i],tat[i]);
    }
    avwt/=i;
    avtat/=i;
    printf("\nnAverage Waiting Time:%d",avwt);
    printf("\nAverage Turnaround Time:%d",avtat);
    return 0;
}
```

Output:

```
┌──(kali㉿kali)-[~/Documents/algorithm]
└─$ ./a.out
Enter total number of processes(maximum 20):3
Enter Process Burst Time
P[1]:12
P[2]:18
P[3]:6

Proces  Burst Time      Waiting Time    Turnaround Time
P[1]            12              0               12
P[2]            18              12              30
P[3]            6               30              36
nAverage Waiting Time:14
Average Turnaround Time:26
```

04. Priority Scheduling Algorithm.

CODE:

```
#include<stdio.h>
struct priority_scheduling
{
    char process_name; int burst_time;
    int waiting_time;
    int turn_around_time; int priority;
};


int main()
{
    int number_of_process; int total = 0;
    struct priority_scheduling temp_process;
    int ASCII_number = 65;
    int position;
    float average_waiting_time;
    float average_turnaround_time;
    printf("Enter the total number of Processes: ");
    scanf("%d", & number_of_process);
    struct priority_scheduling process[number_of_process];
    printf("\nPlease Enter the Burst Time and Priority of each
process:\n");

    for (int i = 0; i < number_of_process; i++)
```

```
    {
        process[i].process_name = (char) ASCII_number;
        printf("\nEnter the details of the process %c \n",
process[i].process_name);
        printf("Enter the burst time: ");
        scanf("%d", & process[i].burst_time);
        printf("Enter the priority: ");
        scanf("%d", & process[i].priority); ASCII_number++;
    }
    for (int i = 0; i < number_of_process; i++)
    {
        position = I;
        for (int j = i + 1; j < number_of_process; j++)
        {
            if (process[j].priority > process[position].priority)
                position = j;
        }
        temp_process = process[i]; process[i] = process[position];
        process[position] = temp_process;
    }
    process[0].waiting_time = 0;
    for (int i = 1; i < number_of_process; i++)
    {
        process[i].waiting_time = 0;
        for (int j = 0; j < i; j++)
        {
            process[i].waiting_time += process[j].burst_time;
        }
        total += process[i].waiting_time;
    }
    average_waiting_time = (float) total / (float)
number_of_process;
    total = 0;

    printf("\n\nProcess_name \t Burst Time \t Waiting Time \t
Turnaround Time\n"); printf(" \n");

    for (int i = 0; i < number_of_process; i++)
    {
        process[i].turn_around_time = process[i].burst_time +
process[i].waiting_time;
        total += process[i].turn_around_time;
```

```
        printf("\t %c \t\t %d \t\t %d \t\t %d",
process[i].process_name, process[i].burst_time,
process[i].waiting_time, process[i].turn_around_time);
        printf("\n  \n");
    }
    average_turnaround_time = (float) total / (float)
number_of_process;
    printf("\n\n Average Waiting Time : %f", average_waiting_time);
    printf("\n Average Turnaround Time: %f\n",
average_turnaround_time);
    return 0;
}
```

Output:

```
┌──(kali㉿kali)-[~/Documents/algorithm]
└─$ ./a.out
Enter the total number of Processes: 3

Please Enter the  Burst Time and Priority of each process:

Enter the details of the process A
Enter the burst time: 12
Enter the priority: 3

Enter the details of the process B
Enter the burst time: 14
Enter the priority: 1

Enter the details of the process C
Enter the burst time: 2
Enter the priority: 2


Process_name     Burst Time      Waiting Time     Turnaround Time
---------------------------------------------------------------
         A              12              0              12
---------------------------------------------------------------
         C              2               12             14
---------------------------------------------------------------
         B              14              14             28
---------------------------------------------------------------


 Average Waiting Time : 8.666667
 Average Turnaround Time: 18.000000
```

THREADS

01. Write a program to execute a thread program.

CODE:

```
#include <pthread.h>
#include <stdio.h>
void* func(void* arg)
```

```c
{
    pthread_detach(pthread_self());
    printf("Inside the thread\n");
    pthread_exit(NULL);
}
void fun()
{
    pthread_t ptid;
    pthread_create(&ptid, NULL, &func, NULL);
    printf("This line may be printed before thread terminates\n");
    if (pthread_equal(ptid, pthread_self()))
        printf("Threads are equal\n");
    else
        printf("Threads are not equal\n");
    pthread_join(ptid, NULL);
    printf("This line will be printed after thread ends\n");
    pthread_exit(NULL);
}
int main()
{
    fun();
    return 0;
}
```

Output:

```
┌──(kali㉿kali)-[~/Documents/thread]
└─$ ./a.out
This line may be printed  before thread terminates
Threads are not equal
Inside the thread
This line will be printed after thread ends
```

02. Write a program to print a message using threads.

CODE:

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

void *Thread(void *vargp)
{
    printf("Program printed using thread \n");
    return NULL;
}


int main()
{
    pthread_t thread_id;
    printf("Before Thread\n");
    pthread_create(&thread_id, NULL,Thread, NULL);
    pthread_join(thread_id, NULL);
    printf("After Thread\n");
    exit(0);
}
```

Output:

```
┌──(kali㉿kali)-[~/Documents/thread]
└─$ ./a.out
Before Thread
Program printed using thread
After Thread
```

03. Write a program to add two numbers using threads.

CODE:

```c
#include <stdio.h>
#include <pthread.h>
int global[2];

void *sum_thread(void *arg)
{
    int *args_array;
    args_array = arg;
```

```
    int n1,n2,sum; n1=args_array[0];
    n2=args_array[1];
    sum = n1+n2;
    printf("N1 + N2 = %d\n",sum);
    return NULL;
}
int main()
{
    printf("First number: ");
    scanf("%d",&global[0]);
    printf("Second number: ");
    scanf("%d",&global[1]);
    pthread_t tid_sum;
    pthread_create(&tid_sum,NULL,sum_thread,global);
    pthread_join(tid_sum,NULL);
    return 0;
}
```

Output:

```
┌──(kali㉿kali)-[~/Documents/thread]
└─$ ./a.out
First number: 10
Second number: 20
10 + 20 = 30
```

04. Write a program to simulate race condition.

CODE:

```
#include<pthread.h>
#include<stdio.h>
#include<unistd.h>
void *fun1();
void *fun2();
int shared=1;
int main()
{
    pthread_t thread1, thread2;
    pthread_create(&thread1, NULL, fun1, NULL);
```

```
    pthread_create(&thread2, NULL, fun2, NULL);
    pthread_join(thread1, NULL);
    pthread_join(thread2,NULL);
    printf("Final value of shared vaiable is %d\n",shared);
}
void *fun1()
{
    int x;
    x=shared;
    printf("Thread1 reads the value of shared variable as %d\n",x);
    x++;
    printf("Local updation by Thread1: %d\n",x); sleep(1);
    shared=x;
    printf("Value of shared variable updated by Thread1 is:
%d\n",shared);
}
void *fun2()
{
    int y;
    y=shared;
    printf("Thread2 reads the value as %d\n",y);
    y--;
    printf("Local updation by Thread2: %d\n",y);
    sleep(1);
    shared=y;
    printf("Value of shared variable updated by Thread2 is:
%d\n",shared);
}
```

Output:

```
┌──(kali㉿kali)-[~/Documents/thread]
└─$ ./a.out
Thread2 reads the value as 1
Local updation by Thread2: 0
Thread1 reads the value of shared variable as 1
Local updation by Thread1: 2
Value of shared variable updated by Thread1 is: 2
Value of shared variable updated by Thread2 is: 0
Final value of shared vaiable is 0
```

05. Write a program to simulate race condition in reverse.

CODE:

```c
#include<pthread.h>
#include<stdo.h>
#include<unistd.h>
int shared=1;
void *fun1()
{
    int x;
    x=shared;
    printf("Thread1 reads the value of shared variable as %d\n",x);
    x--;
    printf("Local updation by Thread1: %d\n",x);
    sleep(1);
    shared=x;
    printf("Value of shared variable updated by Thread1 is:
%d\n",shared);
}
void *fun2()
{
    int y;
    y=shared;
    printf("Thread2 reads the value as %d\n",y);
    y++;
    printf("Local updation by Thread2: %d\n",y);
    sleep(1);
    shared=y;
    printf("Value of shared variable updated by Thread2 is:
%d\n",shared);
}
int main()
{
    pthread_t thread1, thread2;
    pthread_create(&thread1, NULL, fun1, NULL);
    pthread_create(&thread2, NULL, fun2, NULL);
    pthread_join(thread1, NULL);
    pthread_join(thread2,NULL);
    printf("Final value of shared variable is %d\n",shared);
}
```

Output:

```
┌──(kali㉿kali)-[~/Documents/thread]
└─$ ./a.out
Thread2 reads the value as 1
Local updation by Thread2: 2
Thread1 reads the value of shared variable as 1
Local updation by Thread1: 0
Value of shared variable updated by Thread2 is: 2
Value of shared variable updated by Thread1 is: 0
Final value of shared vaiable is 0
```

SEMAPHORE

01. To write a program producer consumer problem using semaphores.

AIM: To Write a C program to simulate producer-consumer problem using semaphores.

DESCRIPTION

Producer consumer problem is a synchronization problem. There is a fixed size buffer where the producer produces items and that is consumed by a consumer process. One solution to the producer- consumer problem uses shared memory. To allow producer and consumer processes to run concurrently, there must be available a buffer of items that can be filled by the producer and emptied by the consumer. This buffer will reside in a region of memory that is shared by the producer and consumer processes. The producer and consumer must be synchronized, so that the consumer does not try to consume an item that has not yet been produced.

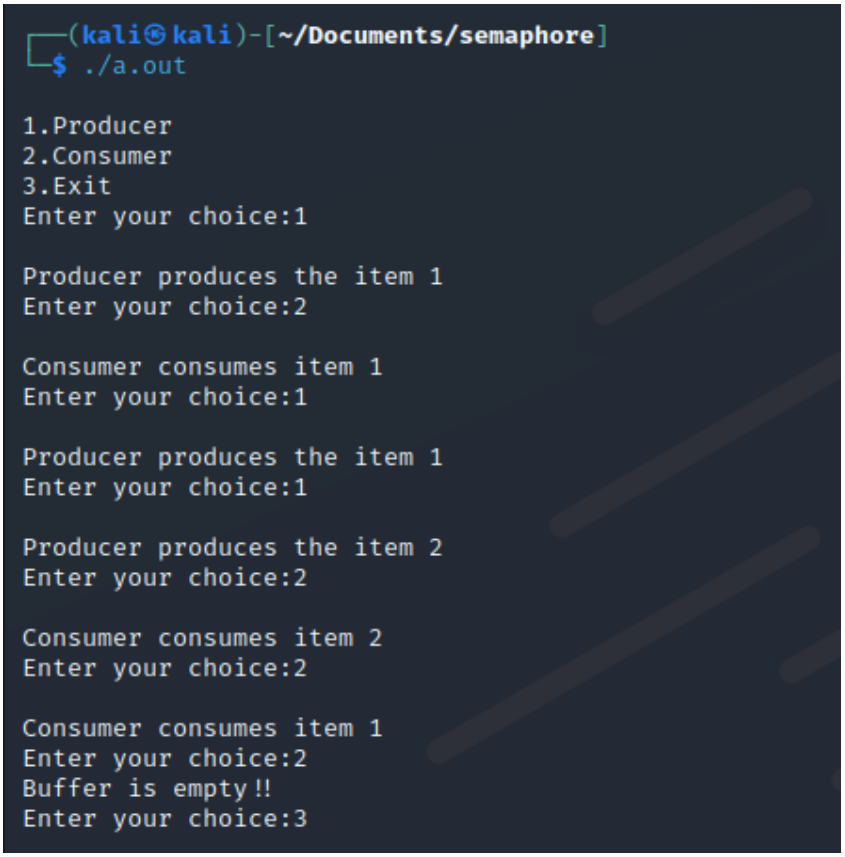
CODE:

```c
#include<stdio.h>
#include<stdlib.h>
int mutex=1,full=0,empty=3,x=0;
void producer();
void consumer();
int main()
{
    int n;
    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);
        }
    }
    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);
}
```

Output:



DEADLOCK AVOIDANCE ALGORITHM

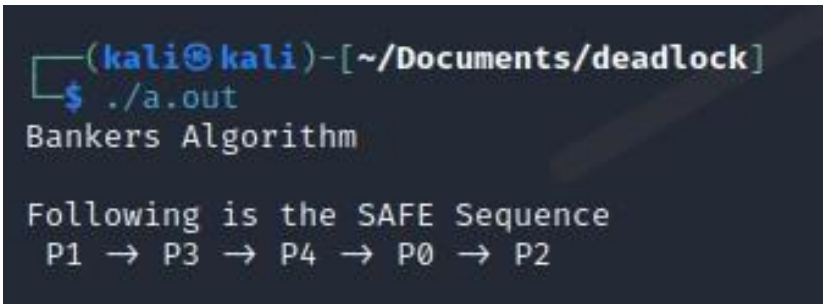
01. To write a program to execute Banker's Algorithm and find the safe sequence.

CODE:

```
#include <stdio.h>
int main()
{

    int n, m, i, j, k; n = 5;
    m = 3;
    int alloc[5][3] = { { 0, 1, 0 },
                        { 2, 0, 0 },
                        { 3, 0, 2 },
                        { 2, 1, 1 },
                        { 0, 0, 2 } };

    int max[5][3] = { { 7, 5, 3 },
                      { 3, 2, 2 },
                      { 9, 0, 2 },
                      { 2, 2, 2 },
                      { 4, 3, 3 } };

    int avail[3] = { 3, 3, 2 };
    int f[n], ans[n], ind = 0;
    for (k = 0; k < n; k++)
    {
        f[k] = 0;
    }
    int need[n][m];
    for (i = 0; i < n; i++)
    {
        for (j = 0; j < m; j++)
            need[i][j] = max[i][j] - alloc[i][j];
    }

    int y = 0;
    for (k = 0; k < 5; k++)
```

```
    {
        for (i = 0; i < n; i++)
        {
            if (f[i] == 0)
            {
                int flag = 0;
                for (j = 0; j < m; j++)
                {
                    if (need[i][j] > avail[j])
                    {
                        flag = 1;
                        break;
                    }
                }
                if (flag == 0)
                {
                    ans[ind++] = i;
                    for (y = 0; y < m; y++)
                        avail[y] += alloc[i][y]; f[i] = 1;
                }
            }
        }
    }
    int flag = 1;
    for(int i=0;i<n;i++)
    {
        if(f[i]==0)
        {
            flag=0;
            printf("The following system is not safe");
            break;
        }
    }
    if(flag==1)
    {
        printf("Following is the SAFE Sequence\n"); for (i = 0; i <
n - 1; i++)
        printf(" P%d ->", ans[i]);
        printf(" P%d", ans[n - 1]);
    }
    return (0);
}
```

Output:


(kali㉿kali)-[~/Documents/deadlock]
$ ./a.out
Bankers Algorithm
Following is the SAFE Sequence
P1 → P3 → P4 → P0 → P2

MEMORY MANAGEMENT

AIM: To Write a program to simulate the following contiguous memory allocation techniques
a) Worst-fit b) Best-fit c) First-fit

DESCRIPTION

One of the simplest methods for memory allocation is to divide memory into several fixed-sized partitions. Each partition may contain exactly one process. In this multiple-partition method, when a partition is free, a process is selected from the input queue and is loaded into the free partition. When the process terminates, the partition becomes available for another process. The operating system keeps a table indicating which parts of memory are available and which are occupied. Finally, when a process arrives and needs memory, a memory section large enough for this process is provided. When it is time to load or swap a process into main memory, and if there is more than one free block of memory of sufficient size, then the operating system must decide which free block to allocate. Best-fit strategy chooses the block that is closest in size to the request. First-fit chooses the first available block that is large enough. Worst-fit chooses the largest available block.

01. To Write a Program to execute First Fit Algorithm.

CODE:

```
#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("\n\t\t\tMemory Management - First Fit\n");
    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++)
        for(j = 0; j < bno; j++)
            if(flags[j] == 0 && bsize[j] >= psize[i])
            {
                allocation[j] = i; flags[j] = 1;
                break;
            }
    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:

```
┌──(kali㉿kali)-[~/Documents/memory]
└─$ ./a.out

                        Memory Management - First Fit
Enter no. of blocks: 3

Enter size of each block: 20 10 30

Enter no. of processes: 3

Enter size of each process: 9 15 19

Block no.       size            process no.             size
1               20              1                       9
2               10              Not allocated
3               30              2                       15
```

02. To Write a Program to execute Best Fit Algorithm.

CODE:

```
#include<stdio.h>
void main()
{
    int fragment[20],b[20],p[20],i,j,nb,np,temp,lowest=9999;
    static int barray[20],parray[20];
    printf("\n\t\t\t Memory Management - Best Fit\n");
    printf("\nEnter the number of blocks:"); scanf("%d",&nb);
    printf("Enter the number of processes:"); scanf("%d",&np);
    printf("\nEnter the size of the blocks:-\n");
    for(i=1;i<=nb;i++)
    {
        printf("Block no.%d:",i);
        scanf("%d",&b[i]);
    }
    printf("\nEnter the size of the processes :-\n");
    for(i=1;i<=np;i++)
    {
        printf("Process no.%d:",i);
        scanf("%d",&p[i]);
    }
    for(i=1;i<=np;i++)
    {
        for(j=1;j<=nb;j++)
        {
            if(barray[j]!=1)
            {
                temp=b[j]-p[i]; if(temp>=0)
                if(lowest>temp)
                {
                    parray[i]=j; lowest=temp;
                }
            }
        }
        fragment[i]=lowest; barray[parray[i]]=1;
        lowest=10000;
    }
```

```
    printf("\nProcess_no\tProcess_size\tBlock_no\tBlock_size\tFragme
nt");
    for(i=1;i<=np && parray[i]!=0;i++)
        printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d",i,p[i],parray[i],b[par
ray[i]],fragment[i]);
}
```

Output:

```
┌──(kali㉿kali)-[~/Documents/memory]
└─$ ./a.out

                        Memory Management -  Best Fit

Enter the number of blocks:3
Enter the number of processes:3

Enter the size of the blocks:-
Block no.1:200
Block no.2:100
Block no.3:250

Enter the size of the processes :-
Process no.1:89
Process no.2:199
Process no.3:201

Process_no      Process_size    Block_no        Block_size      Fragment
1               89              2               100             11
2               199             1               200             1
3               201             3               250             49
```

03. To Write a program to execute Worst Fit Algorithm.

CODE:

```
#include<stdio.h>
int main()
{
    int n,n1,i;
    printf("\n\t\t\tMemory Management - Worst Fit\n\n");
    printf("Enter the number of processes:");
    scanf("%d",&n);
    int process[n];
    printf("\n Enter the size of processes:\n");
    for(i=0;i<n;i++)
    {
        scanf("%d",&process[i]);
    }
    printf("Enter the no of memoryblocks:");
    scanf("%d",&n1);
    int blocks[n1];
    printf("\n Enter the size of blocks:\n");
    int total=0;
    for(i=0;i<n1;i++)
    {
```

```
        scanf("%d",&blocks[i]); total=total+blocks[i];
    }
    int process1[n1];
    int job[n1];
    int frag[n1];
    int check[n1];
    for(i=0;i<n1;i++)
    {
        check[i]=0;
    }
    int j,used=0; i=0;
    while(i<n)
    {
        int max=-1,j1=-1,k=-1,max1;
        for(j=0;j<n1;j++)
        {
            max1=blocks[j];
            if(max1>=max&&check[j]==0&&max1>=process[i])
            {
                max=max1; j1=j;
            }
            else
            {
                if(check[j]==0)
                {
                    process1[j]=0; job[j]=0; frag[j]=blocks[j];
                }
            }
        }
        if(k!=j1)
        {
            process1[j1]=process[i]; job[j1]=i+1;
            frag[j1]=blocks[j1]-process[i];
            used=used+process[i];
            check[j1]=1;
            int l;
        }
        i++;
    }
    printf("block_size\tprocess_size\tprocess_no\tfragmentation\n");
    for(i=0;i<n1;i++)
    {
```

```
        printf("%d\t\t%d\t\t%d\t\t%d\n",blocks[i],process1[i],job[i]
,frag[i]);
    }
}
```

Output:

```
┌──(kali㉿kali)-[~/Documents/memory]
└─$ ./a.out

                        Memory Management - Worst Fit

Enter the number of processes:2

 Enter the size of processes:
10 20
Enter the no of memoryblocks:2

 Enter the size of blocks:
10 20
block_size      process_size    process_no      fragmentation
10              0               0               10
20              10              1               10
```

PAGE – REPLACEMENT ALGORITHMS

AIM: To implement FIFO page replacement technique.

DESCRIPTION:

Page replacement algorithms are an important part of virtual memory management and it helps the OS to decide which memory page can be moved out making space for the currently needed page. However, the ultimate objective of all page replacement algorithms is to reduce the number of page faults.

FIFO-This is the simplest page replacement algorithm. In this algorithm, the operating system keeps track of all pages in the memory in a queue, the oldest page is in the front of the queue. When a page needs to bereplaced page in the front of the queue is selected for removal.

LRU-In this algorithm page will be replaced which is least recently used

OPTIMAL- In this algorithm, pages are replaced which would not be used for the longest duration of time in the future. This algorithm will give us less page fault when compared to other page replacement algorithms.

ALGORITHM:

1. Start the process
2. Read number of pages n
3. Read number of pages no
4. Read page numbers into an array a[i]
5. Initialize avail[i]=0 .to check page hit
6. Replace the page with circular queue, while re-placing check page availability in the

   framePlace avail[i]=1 if page is placed in the frame Count page faults
7. Print the results.
8. Stop the process.

FIRST IN FIRST OUT

01. To write a program to execute First in First out Page Replacement Algorithm.

CODE:

```
#include <stdio.h>
int main()
{
    int incomingStream[] = {4, 1, 2, 4, 5};
```

```
    int pageFaults = 0;
    int frames = 3;
    int m, n, s, pages;
    pages = sizeof(incomingStream)/sizeof(incomingStream[0]);
    printf("Incoming \t Frame 1 \t Frame 2 \t Frame 3");
    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(incomingStream[m] == temp[n])
            {
                s++;
                pageFaults--;
            }
        }
        pageFaults++;
        if((pageFaults <= frames) && (s == 0))
        {
            temp[m] = incomingStream[m];
        }
        else if(s == 0)
        {
            temp[(pageFaults - 1) % frames] = incomingStream[m];
        }
        printf("\n"); printf("%d\t\t\t",incomingStream[m]);
        for(n = 0; n < frames; n++)
        {
            if(temp[n] != -1)
                printf(" %d\t\t\t", temp[n]);
            else
                printf(" - \t\t\t");
        }
    }
    printf("\nTotal Page Faults:\t%d\n", pageFaults);
    return 0;
}
```

Output:

```
┌──(kali㉿kali)-[~/Documents/pagereplace]
└─$ ./a.out
Incoming        Frame 1         Frame 2         Frame 3
4                       4                       -                       -
1                       4                       1                       -
2                       4                       1                       2
4                       4                       1                       2
5                       5                       1                       2
Total Page Faults:      4
```

02. To write a program to execute Least Recently Used Page Replacement Algorithm.

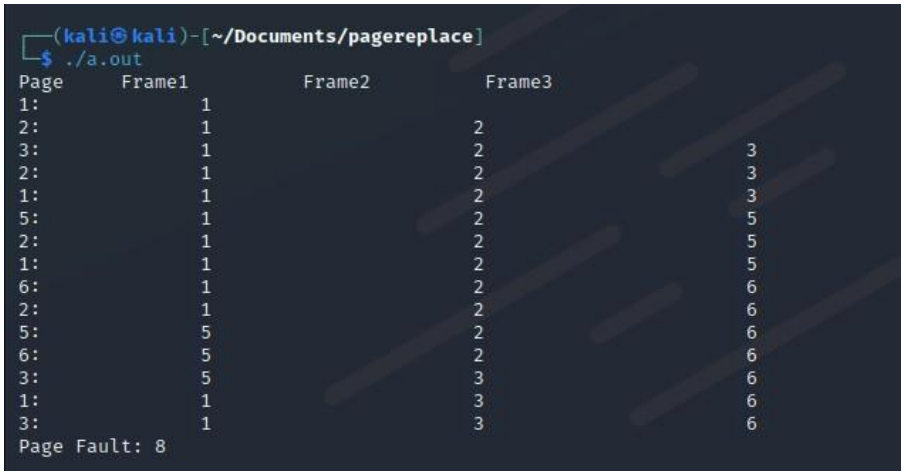
CODE:

```
#include<stdio.h>
#include<limits.h>
int checkHit(int incomingPage, int queue[], int occupied)
{
    for(int i = 0; i < occupied; i++)
    {
        if(incomingPage == queue[i])
            return 1;
    }
    return 0;
}
void printFrame(int queue[], int occupied)
{
    for(int i = 0; i < occupied; i++)
        printf("%d\t\t\t",queue[i]);
}
int main()
{
    int incomingStream[] = {1, 2, 3, 2, 1, 5, 2, 1, 6, 2, 5, 6, 3,
1, 3};
    int n = sizeof(incomingStream)/sizeof(incomingStream[0]);
    int frames = 3;
    int queue[n];
    int distance[n];
    int occupied = 0;
    int pagefault = 0;
    printf("Page\t Frame1 \t Frame2 \t Frame3\n");
    for(int i = 0;i < n; i++)
```

```
    {
        printf("%d: \t\t",incomingStream[i]);
        if(checkHit(incomingStream[i], queue, occupied))
        {
            printFrame(queue, occupied);
        }
        else if(occupied < frames)
        {
            queue[occupied] = incomingStream[i];
            pagefault++;
            occupied++;
            printFrame(queue, occupied);
        }
        else
        {
            int max = INT_MIN;
            int index;
            for (int j = 0; j < frames; j++)
            {
                distance[j] = 0;
                // traverse in reverse direction to find

                // at what distance frame item occurred last
                for(int k = i - 1; k >= 0; k--)
                {
                    ++distance[j];
                    if(queue[j] == incomingStream[k]);
                    break;
                }
                if(distance[j] > max)
                {
                    max = distance[j]; index = j;
                }
            }
            queue[index] = incomingStream[i]; printFrame(queue,
occupied);
            pagefault++;
        }
        printf("\n");
    }
    printf("Page Fault: %d",pagefault);
    return 0;
```

```
}
```

Output:



05. To write a program to execute Optimal Page Replacement Algorithm.

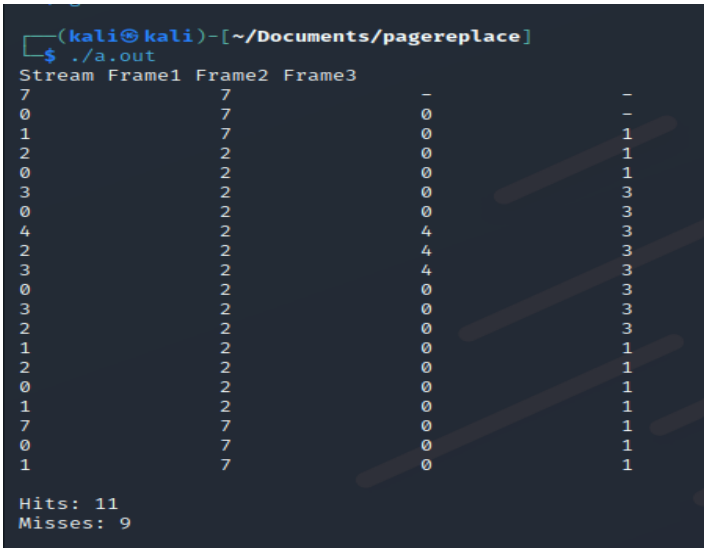
CODE:

```
#include <stdio.h>
int search(int key, int frame_items[], int frame_occupied)
{
    for (int i = 0; i < frame_occupied; i++)
        if (frame_items[i] == key)
        return 1;
    return 0;
}

void printOuterStructure(int max_frames)
{
    printf("Stream ");
    for(int i = 0; i < max_frames; i++)
    printf("Frame%d ", i+1);
}
void printCurrFrames(int item, int frame_items[], int
frame_occupied, int max_frames)
{
    printf("\n%d \t\t", item);
    for(int i = 0; i < max_frames; i++)
    {
```

```
        if(i < frame_occupied)
            printf("%d \t\t", frame_items[i]);
        else
            printf("- \t\t");
    }
}
int predict(int ref_str[], int frame_items[], int refStrLen, int
index, int frame_occupied)
{
    int result = -1, farthest = index;
    for (int i = 0; i < frame_occupied; i++)
    {
        int j;
        for (j = index; j < refStrLen; j++)
        {
            if (frame_items[i] == ref_str[j])
            {
                if (j > farthest)
                {
                    farthest = j; result = i;
                }
                break;
            }
        }
        if (j == refStrLen)
            return i;
    }
    return (result == -1) ? 0 : result;
}

void optimalPage(int ref_str[], int refStrLen, int frame_items[],
int max_frames)
{
    int frame_occupied = 0;

    printOuterStructure(max_frames);
    int hits = 0;
    for (int i = 0; i < refStrLen; i++)
    {
        if (search(ref_str[i], frame_items, frame_occupied))
        {
            hits++;
```

```
            printCurrFrames(ref_str[i], frame_items, frame_occupied,
max_frames);
            continue;
        }
        if (frame_occupied < max_frames)
        {
            frame_items[frame_occupied] = ref_str[i];
            frame_occupied++;
            printCurrFrames(ref_str[i], frame_items, frame_occupied,
max_frames);
        }
        else
        {
            int pos = predict(ref_str, frame_items, refStrLen, i +
1, frame_occupied);
            frame_items[pos] = ref_str[i];
            printCurrFrames(ref_str[i], frame_items, frame_occupied,
max_frames);
        }
    }
    printf("\n\nHits: %d\n", hits);
    printf("Misses: %d", refStrLen - hits);
}
int main()
{
    // int ref_str[] = {9, 0, 5, 1, 0, 3, 0, 4, 1, 3, 0, 3, 1, 3};
    int ref_str[] = {7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0,
1, 7, 0, 1};
    int refStrLen = sizeof(ref_str) / sizeof(ref_str[0]); int
max_frames = 3;
    int frame_items[max_frames];

    optimalPage(ref_str, refStrLen, frame_items, max_frames); return
0;
}
```

Output:



```
┌──(kali㉿kali)-[~/Documents/pagereplace]
└─$ ./a.out
Stream Frame1 Frame2 Frame3
7               7               -               -
0               7               0               -
1               7               0               1
2               2               0               1
0               2               0               1
3               2               0               3
0               2               0               3
4               2               4               3
2               2               4               3
3               2               4               3
0               2               0               3
3               2               0               3
2               2               0               3
1               2               0               1
2               2               0               1
0               2               0               1
1               2               0               1
7               7               0               1
0               7               0               1
1               7               0               1

Hits: 11
Misses: 9
```

PAGING MEMORY MANAGEMENT

01. To write a program to simulate paging technique of memory management.

CODE:

```
#include<stdio.h>
#define MAX 50
int main()
{
    int page[MAX],i,n,f,ps,off,pno;
    int choice=0;
    printf("\nEnter the no of peges in memory: ");
    scanf("%d",&n);
    printf("\nEnter page size: ");
    scanf("%d",&ps);
    printf("\nEnter no of frames: ");
    scanf("%d",&f);
    for(i=0;i<n;i++)
        page[i]=-1;
    printf("\nEnter the page table\n");
    printf("(Enter frame no as -1 if that page is not present in any
frame)\n\n");
    printf("\npageno\tframeno\n-------\t    ");
    for(i=0;i<n;i++)
    {
        printf("\n\n%d\t\t",i); scanf("%d",&page[i]);
    }
    do
    {
        printf("\n\nEnter the logical address(i.e,page no &
offset):");
        scanf("%d%d",&pno,&off);
        if(page[pno]==-1)
            printf("\n\nThe required page is not available in any of
frames");
        else
            printf("\n\nPhysical address(i.e,frame no &
offset):%d,%d",page[pno],off);
        printf("\nDo you want to continue(1/0)?:");
        scanf("%d",&choice);
    }while(choice==1);
```

```
    return 1;
}
```

Output: