PROGRAM NO.: 01

Write a C program to implement the I/O system calls of UNIX/LINUX operating system. (open (), close (), read (), write (), fork ())

open()

```
/ C program to illustrate
// open system call
#include <errno.h>
#include <fcntl.h>
#include <stdio.h>
#include <unistd.h>

extern int errno;
int main()
{
   // if file does not have in directory
   // then file foo.txt is created.
int fd = open("foo.txt", O_RDONLY | O_CREAT);
printf("fd = %d\n", fd);
if (fd == -1) {
       // print which type of error have in a code
   printf("Error Number % d\n", errno);
 //print program detail "Success or failure"
     perror("Program");
   }
   return 0;
}
```

close()

```
/ C program to illustrate close system Call
#include <fcntl.h>
#include <stdio.h>
#include <unistd.h>

int main()
{
   int fd1 = open("foo.txt", O_RDONLY);
   if (fd1 < 0) {
      perror("c1");
      exit(1);
   }
   printf("opened the fd = % d\n", fd1);
   // Using close system Call
   if (close(fd1) < 0) {
      perror("c1");
      exit(1);
   }
   printf("closed the fd.\n");
}
```

read()

```
#include <unistd.h>
int main()
{
   int fd, sz;
   char* c = (char*)calloc(100, sizeof(char));

   fd = open("foo.txt", O_RDONLY);
   if (fd < 0) {
      perror("r1");
      exit(1);
   }

   sz = read(fd, c, 10);
   printf("called read(% d, c, 10). returned that"
         " %d bytes were read.\n",
         fd, sz);
   c[sz] = '\0';
   printf("Those bytes are as follows: % s\n", c);

   return 0;
}
```

write()

```
// C program to illustrate
// write system Call
#include<stdio.h>
#include <fcntl.h>
main()
{
int sz;

int fd = open("foo.txt", O_WRONLY | O_CREAT | O_TRUNC, 0644);
if (fd < 0)
{
   perror("r1");
   exit(1);
}

sz = write(fd, "hello skillvertex\n", strlen("hello skillvertex\n"));
printf("called write(% d, \"hello skillvertex\\n\", %d)."
   " It returned %d\n", fd, strlen("hello skillvertex\n"), sz);
  close(fd);
}
```

fork()

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
int main()
```

```
{

        // make two process which run same
        // program after this instruction
        pid_t p = fork();
        if(p<0){
        perror("fork fail");
        exit(1);
        }
        printf("Hello world!, process_id(pid) = %d \n",getpid());
        return 0;
}
```

Example: Print "hello world" from the program without using any printf function.

```
// C program to illustrate
// I/O system Calls
#include <fcntl.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>

int main(void)
{
   int fd[2];
   char buf1[12] = "hello world";
   char buf2[12];

   // assume foobar.txt is already created
   fd[0] = open("foobar.txt", O_RDWR);
   fd[1] = open("foobar.txt", O_RDWR);

   write(fd[0], buf1, strlen(buf1));
   write(1, buf2, read(fd[1], buf2, 12));

   close(fd[0]);
   close(fd[1]);

   return 0;
}
```

PROGRAM NO. : 02

Write a C program to simulate the following CPU scheduling Algorithms.

a) FCFS b) SJF c) Round Robin

a) FCFS

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

OUTPUT

```
Enter the number of processes -- 3

Enter Burst Time for Process 0 -- 24

Enter Burst Time for Process 1 -- 3

Enter Burst Time for Process 2 -- 3
        PROCESS     BURST TIME          WAITING TIME       TURNAROUND TIME

        P0              24              0               24
        P1              3               24              27
        P2              3               27              30
Average Waiting Time -- 17.000000
Average Turnaround Time -- 27.000000
```

b) SJF

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

OUTPUT

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

PROCESS      BURST TIME          WAITING TIME       TURNAROUND TIME

        P3              3               0               3
```

```
        P0          6           3           9
        P2          7           9           16
        P1          8           16          24
Average Waiting Time -- 7.000000
Average Turnaround Time -- 13.000000

=== Code Execution Successful ===
```

c) ROUND ROBIN SCHEDULING

```
#include<stdio.h>
int main()
{
   //Input no of processed
   int  n;
   printf("Enter Total Number of Processes:");
   scanf("%d", &n);
   int wait_time = 0, ta_time = 0, arr_time[n], burst_time[n], temp_burst_time[n];
   int x = n;

   //Input details of processes
   for(int i = 0; i < n; i++)
   {
      printf("Enter Details of Process %d \n", i + 1);
      printf("Arrival Time:  ");
      scanf("%d", &arr_time[i]);
      printf("Burst Time:   ");
      scanf("%d", &burst_time[i]);
      temp_burst_time[i] = burst_time[i];
   }

   //Input time slot
   int time_slot;
   printf("Enter Time Slot:");
   scanf("%d", &time_slot);

   //Total indicates total time
   //counter indicates which process is executed
   int total = 0,  counter = 0,i;
   printf("Process ID      Burst Time      Turnaround Time      Waiting Time\n");
   for(total=0, i = 0; x!=0; )
   {
      // define the conditions
      if(temp_burst_time[i] <= time_slot && temp_burst_time[i] > 0)
      {
         total = total + temp_burst_time[i];
         temp_burst_time[i] = 0;
         counter=1;
      }
      else if(temp_burst_time[i] > 0)
      {
```

```
            temp_burst_time[i] = temp_burst_time[i] - time_slot;
            total  += time_slot;
        }
        if(temp_burst_time[i]==0 && counter==1)
        {
            x--; //decrement the process no.
            printf("\nProcess No %d  \t\t %d\t\t\t\t %d\t\t\t\t %d", i+1, burst_time[i],
                  total-arr_time[i], total-arr_time[i]-burst_time[i]);
            wait_time = wait_time+total-arr_time[i]-burst_time[i];
            ta_time += total -arr_time[i];
            counter =0;
        }
        if(i==n-1)
        {
            i=0;
        }
        else if(arr_time[i+1]<=total)
        {
            i++;
        }
        else
        {
            i=0;
        }
    }
    float average_wait_time = wait_time * 1.0 / n;
    float average_turnaround_time = ta_time * 1.0 / n;
    printf("\nAverage Waiting Time:%f", average_wait_time);
    printf("\nAverage Turnaround Time:%f", average_turnaround_time);
    return 0;
}
```

OUTPUT

```
Enter Total Number of Processes:3
Enter Details of Process 1
Arrival Time:  0
Burst Time:   10
Enter Details of Process 2
Arrival Time:  1
Burst Time:   8
Enter Details of Process 3
Arrival Time:  2
Burst Time:   7
Enter Time Slot:5
Process ID      Burst Time      Turnaround Time      Waiting Time

Process No 1            10                      20                      10
Process No 2            8                       22                      14
Process No 3            7                       23                      16
Average Waiting Time:13.333333
```

Average Turnaround Time:21.666666

=== Code Execution Successful ===

PROGRAM NO.: 03

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

```
#include <iostream>
using namespace std;
int main()
{
   // P0, P1, P2, P3, P4 are the Process names here
  int n, m, i, j, k;
  n = 5; // Number of processes
  m = 3; // Number of resources
  int alloc[5][3] = { { 0, 1, 0 }, // P0 // Allocation Matrix
                      { 2, 0, 0 }, // P1
                      { 3, 0, 2 }, // P2
                      { 2, 1, 1 }, // P3
                      { 0, 0, 2 } }; // P4

  int max[5][3] = { { 7, 5, 3 }, // P0 // MAX Matrix
                    { 3, 2, 2 }, // P1
                    { 9, 0, 2 }, // P2
                    { 2, 2, 2 }, // P3
                    { 4, 3, 3 } }; // P4

  int avail[3] = { 3, 3, 2 }; // Available Resources

  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;

// To check if sequence is safe or not
for(int i = 0;i<n;i++)
{
     if(f[i]==0)
    {
      flag = 0;
      cout << "The given sequence is not safe";
      break;
    }
}

if(flag==1)
{
  cout << "Following is the SAFE Sequence" << endl;
    for (i = 0; i < n - 1; i++)
      cout << " P" << ans[i] << " ->";
    cout << " P" << ans[n - 1] <<endl;
}

  return (0);
}
```

OUTPUT

```
Following is the SAFE Sequence
 P1 -> P3 -> P4 -> P0 -> P2


=== Code Execution Successful ===
```

PROGRAM NO.: 04

Write a C program to illustrate the Inter Process Communication (IPC) mechanism using Message Queues.

MESSAGE QUEUE FOR WRITER PROCESS

```
// C Program for Message Queue (Writer Process)
#include <stdio.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#define MAX 100

// structure for message queue
struct mesg_buffer {
        long mesg_type;
        char mesg_text[100];
} message;

int main()
{
        key_t key;
        int msgid;

        // ftok to generate unique key
        key = ftok("progfile", 65);

        // msgget creates a message queue
        // and returns identifier
        msgid = msgget(key, 0666 | IPC_CREAT);
        message.mesg_type = 1;

        printf("Write Data : ");
        fgets(message.mesg_text,MAX,stdin);

        // msgsnd to send message
        msgsnd(msgid, &message, sizeof(message), 0);

        // display the message
        printf("Data send is : %s \n", message.mesg_text);

        return 0;
}
```

OUTPUT

```
Write Data : WELCOME TO OS LAB
Data send is : WELCOME TO OS LAB

=== Code Execution Successful ===
```

MESSAGE QUEUE FOR READER PROCESS

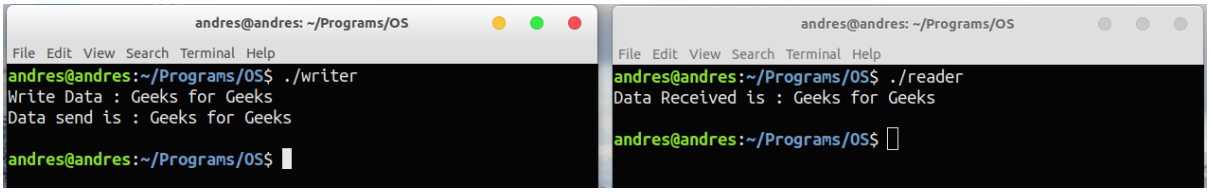
```
#include <stdio.h>
#include <sys/ipc.h>
#include <sys/msg.h>
// structure for message queue
struct mesg_buffer {
long mesg_type;
char mesg_text[100];
} message;
int main()
{
key_t key;
int msgid;
// ftok to generate unique key
key = ftok("progfile", 65);
// msgget creates a message queue
// and returns identifier
msgid = msgget(key, 0666 | IPC_CREAT);
// msgrcv to receive message
msgrcv(msgid, &message, sizeof(message), 1, 0);
// display the message
printf("Data Received is : %s \n",
message.mesg_text);
// to destroy the message queue
msgctl(msgid, IPC_RMID, NULL);
return 0;
}
```

OUTPUT

Data Received is: WELCOME TO OS LAB

=== Code Execution Successful ===

SAMPLE OUTPUT:



PROGRAM NO.: 05

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

```
// C program for the above approach
#include <stdio.h>
#include <stdlib.h>

// Initialize a mutex to 1
int mutex = 1;

// Number of full slots as 0
int full = 0;

// Number of empty slots as size
// of buffer
int empty = 10, x = 0;

// Function to produce an item and
// add it to the buffer
void producer()
{
    // Decrease mutex value by 1
    --mutex;

    // Increase the number of full
    // slots by 1
    ++full;

    // Decrease the number of empty
    // slots by 1
    --empty;

    // Item produced
    x++;
    printf("\nProducer produces"
           "item %d",
           x);

    // Increase mutex value by 1
    ++mutex;
}

// Function to consume an item and
// remove it from buffer
void consumer()
{
    // Decrease mutex value by 1
    --mutex;
```

```
    // Decrease the number of full
    // slots by 1
    --full;

    // Increase the number of empty
    // slots by 1
    ++empty;
    printf("\nConsumer consumes ""item %d",x);
    x--;

    // Increase mutex value by 1
    ++mutex;
}

// Driver Code
int main()
{
    int n, i;
    printf("\n1. Press 1 for Producer""\n2. Press 2 for Consumer""\n3. Press 3 for Exit");

    for (i = 1; i > 0; i++) {

        printf("\nEnter your choice:");
        scanf("%d", &n);

        // Switch Cases
        switch (n) {
        case 1:

            // If mutex is 1 and empty
            // is non-zero, then it is
            // possible to produce
            if ((mutex == 1)
                && (empty != 0)) {
                producer();
            }

            // Otherwise, print buffer
            // is full
            else {
                printf("Buffer is full!");
            }
            break;

        case 2:

            // If mutex is 1 and full
            // is non-zero, then it is
            // possible to consume
            if ((mutex == 1)
```

```
            && (full != 0)) {
            consumer();
        }

        // Otherwise, print Buffer
        // is empty
        else {
            printf("Buffer is empty!");
        }
        break;

    // Exit Condition
    case 3:
        exit(0);
        break;
    }
  }
}
```

OUTPUT

```
1. Press 1 for Producer
2. Press 2 for Consumer
3. Press 3 for Exit
Enter your choice:2
Buffer is empty!
Enter your choice:1

Producer producesitem 1
Enter your choice:1

Producer producesitem 2
Enter your choice:1

Producer producesitem 3
Enter your choice:2

Consumer consumes item 3
Enter your choice:1

Producer producesitem 3
Enter your choice:2

Consumer consumes item 3
Enter your choice:2

Consumer consumes item 2
Enter your choice:2

Consumer consumes item 1
```

```
Enter your choice:2
Buffer is empty!
Enter your choice:3
=== Code Execution Successful ===
```

PROGRAM NO.: 06

Write a C program to simulate the following contiguous memory allocation techniques

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

a)WORST-FIT

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

```
}
printf("\nFile_no  File_size  Block_no  Block_size  Fragement");
for(i=1;i<=nf;i++)
printf("\n%d\t\t\t%d\t\t\t%d\t\t\t%d\t\t\t%d",i,f[i],ff[i],b[ff[i]],frag[i]);
}
```

OUTPUT

```
Memory Management Scheme - First Fit
Enter the number of blocks:3
Enter the number of files:2

Enter the size of the blocks:-
Block 1:5
Block 2:2
Block 3:7
Enter the size of the files :-
File 1:1
File 2:4
```

File_no	File_size	Block_no	Block_size	Fragement
1	1	1	5	4
2	4	3	7	3

b) Best-fit

```
#include<stdio.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:3
Enter the number of files:2

Enter the size of the blocks:-
Block 1:5
Block 2:2
Block 3:7
Enter the size of the files :-
File 1:1
File 2:4

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

c) First-fit

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

```
scanf("%d",&b[i]);
}
printf("Enter the size of the files :-\n");
for(i=1;i<=nf;i++)
{
printf("File %d:",i);
scanf("%d",&f[i]);
}
for(i=1;i<=nf;i++)
{

for(j=1;j<=nb;j++)
{
if(bf[j]!=1) //if bf[j] is not allocated
{
temp=b[j]-f[i];
if(temp>=0)
if(highest<temp)
{
ff[i]=j;
highest=temp;
}
}
}
frag[i]=highest;
bf[ff[i]]=1;
highest=0;
}
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]);
}
```

OUTPUT

```
    Memory Management Scheme - Worst Fit
Enter the number of blocks:3
Enter the number of files:2

Enter the size of the blocks:-
Block 1:5
Block 2:2
Block 3:7
Enter the size of the files :-
File 1:1
File 2:4

File_no:    File_size :    Block_no:    Block_size:    Fragement
1           1              3            7              6
2           4              1            5              1
```

PROGRAM NO.: 07

Write C programs to simulate Page Replacement Algorithms: a) FIFO, b) LRU.

a) FIFO

```
#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

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
=== Code Execution Successful ===
```

b) LRU

```
#include<stdio.h>
int main()
{
int q[20],p[50],c=0,c1,d,f,i,j,k=0,n,r,t,b[20],c2[20];
printf("Enter no of pages:");
scanf("%d",&n);
printf("Enter the reference string:");
for(i=0;i<n;i++)
        scanf("%d",&p[i]);
printf("Enter no of frames:");
scanf("%d",&f);
q[k]=p[k];
printf("\n\t%d\n",q[k]);
c++;
k++;
for(i=1;i<n;i++)
        {
                c1=0;
                for(j=0;j<f;j++)
                {
                        if(p[i]!=q[j])
                        c1++;
                }
                if(c1==f)
                {
                        c++;
                        if(k<f)
                        {
                                q[k]=p[i];
                                k++;
                                for(j=0;j<k;j++)
                                printf("\t%d",q[j]);
                                printf("\n");
                        }
                        else
                        {
```

```
                        for(r=0;r<f;r++)
                        {
                                c2[r]=0;
                                for(j=i-1;j<n;j--)
                                {
                                if(q[r]!=p[j])
                                c2[r]++;
                                else
                                break;
                        }
                }
                for(r=0;r<f;r++)
                 b[r]=c2[r];
                for(r=0;r<f;r++)
                {
                        for(j=r;j<f;j++)
                        {
                                if(b[r]<b[j])
                                {
                                        t=b[r];
                                        b[r]=b[j];
                                        b[j]=t;
                                }
                        }
                }
                for(r=0;r<f;r++)
                {
                        if(c2[r]==b[0])
                        q[r]=p[i];
                        printf("\t%d",q[r]);
                }
                printf("\n");
        }
    }
}
printf("\nThe no of page faults is %d",c);
}
```

OUTPUT

```
Enter no of pages:10
Enter the reference string:7 5 9 4 3 7 9 6 2 1
Enter no of frames:3

    7
    7       5
    7       5       9
    4       5       9
    4       3       9
    4       3       7
    9       3       7
```

```
9       6       7
9       6       2
1       6       2
```

The no of page faults is 10

=== Code Execution Successful ===

PROGRAM NO.: 08
Write C program to implement Linked File Allocation Method.

```
#include<stdio.h>
#include<stdlib.h>
void main()
{
int f[50], p,i, st, len, j, c, k, a;
for(i=0;i<50;i++)
f[i]=0;
printf("Enter how many blocks already allocated: ");
scanf("%d",&p);
printf("Enter blocks already allocated: ");
for(i=0;i<p;i++)
{
scanf("%d",&a);
f[a]=1;
}
x: printf("Enter index starting block and length: ");
scanf("%d%d", &st,&len);
k=len;
if(f[st]==0)
{
for(j=st;j<(st+k);j++)
{
if(f[j]==0)
{
f[j]=1;
printf("%d-------->%d\n",j,f[j]);
}
else
{
printf("%d Block is already allocated \n",j);
k++;
}
}
}
else
printf("%d starting block is already allocated \n",st);
printf("Do you want to enter more file(Yes - 1/No - 0)");
scanf("%d", &c);
```

```
if(c==1)
goto x;
else
exit(0);
}
```

OUTPUT

```
Enter how many blocks already allocated: 3
Enter blocks already allocated: 1 3 5
Enter index starting block and length: 2 2
2-------->1
3 Block is already allocated
4-------->1
Do you want to enter more file(Yes - 1/No - 0)0


=== Code Execution Successful ===
```