EXPERIMENT-1

AIM: Write a program to implement CPU scheduling for first come first serve.

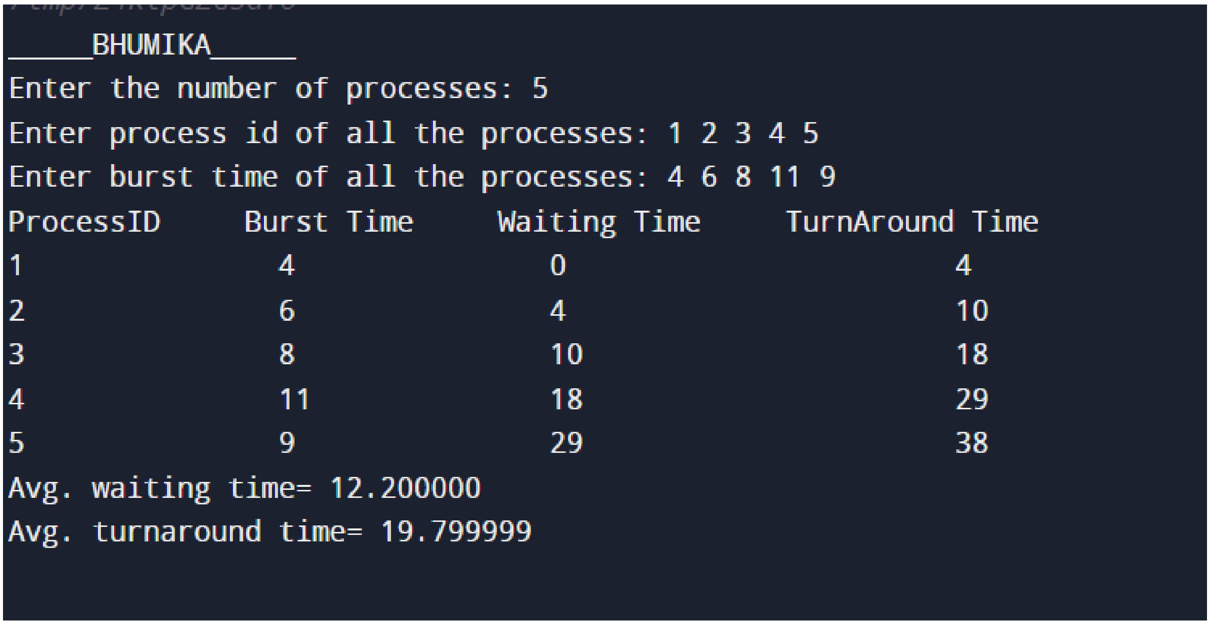
PROGRAM CODE:

```
#include <stdio.h>
int main()
{
   int pid[15];
   int bt[15];
   printf("_____BHUMIKA_____\n");
   int n;
   printf("Enter the number of processes: ");
   scanf("%d",&n);

   printf("Enter process id of all the processes: ");
   for(int i=0;i<n;i++)
   {
      scanf("%d",&pid[i]);
   }
printf("Enter burst time of all the processes: ");
   for(int i=0;i<n;i++)
   {
      scanf("%d",&bt[i]);
   }
```

```
int i, wt[n];
wt[0]=0;

for(i=1; i<n; i++)
{
   wt[i]= bt[i-1]+ wt[i-1];
}
printf("ProcessID     Burst Time     Waiting Time     TurnAround Time\n");
float twt=0.0;
float tat= 0.0;
for(i=0; i<n; i++)
{
   printf("%d\t\t\t\t", pid[i]);
   printf("%d\t\t\t\t", bt[i]);
   printf("%d\t\t\t\t\t\t", wt[i]);
   printf("%d\t\t\t\t\t\t\t", bt[i]+wt[i]);
   printf("\n");
   twt += wt[i];
   tat += (wt[i]+bt[i]);
}
float att,awt;
awt = twt/n;
```

```
    att = tat/n;

    printf("Avg. waiting time= %f\n",awt);

    printf("Avg. turnaround time= %f",att);

}
```

OUTPUT:



```
_____BHUMIKA_____
Enter the number of processes: 5
Enter process id of all the processes: 1 2 3 4 5
Enter burst time of all the processes: 4 6 8 11 9
ProcessID     Burst Time     Waiting Time     TurnAround Time
1               4               0                       4
2               6               4                       10
3               8               10                      18
4               11              18                      29
5               9               29                      38
Avg. waiting time= 12.200000
Avg. turnaround time= 19.799999
```

EXPERIMENT-2

AIM: Write a program to implement CPU scheduling for shortest job first.

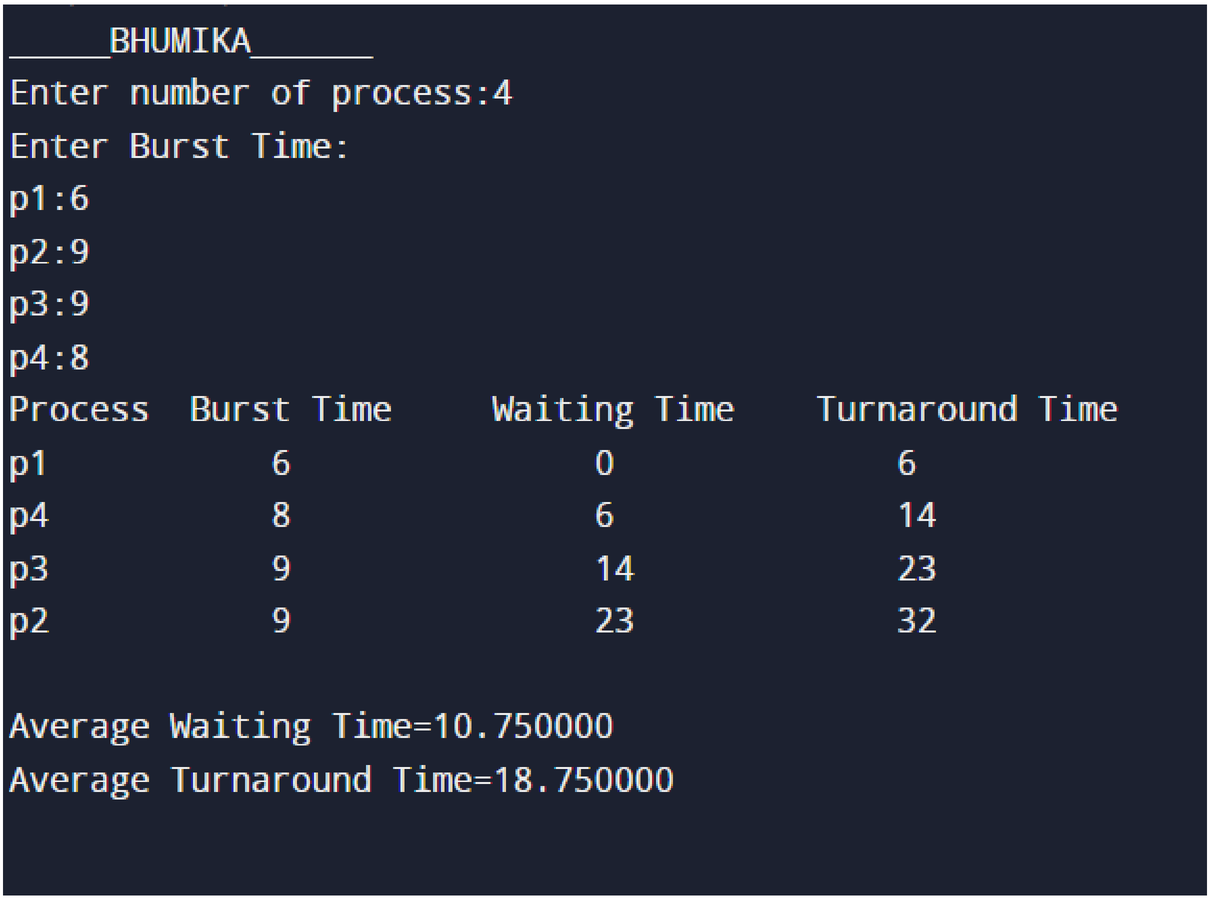
PROGRAM CODE:

```
#include<stdio.h>
int main()
{
   int bt[20],p[20],wt[20],tat[20],i,j,n,total=0,totalT=0,pos,temp;
   float avg_wt,avg_tat;
   printf("_____BHUMIKA______\n");
   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;
 printf("\nProcess\t Burst Time \tWaiting Time\tTurnaround Time");
for(i=0;i<n;i++)
```

```
    {
       tat[i]=bt[i]+wt[i];
       totalT+=tat[i];
       printf("\np%d\t\t\t %d\t\t\t\t %d\t\t\t\t%d",p[i],bt[i],wt[i],tat[i]);
    }
    avg_tat=(float)totalT/n;
    printf("\n\nAverage Waiting Time=%f",avg_wt);
    printf("\nAverage Turnaround Time=%f",avg_tat);
}
```

OUTPUT:



```
_____BHUMIKA______
Enter number of process:4
Enter Burst Time:
p1:6
p2:9
p3:9
p4:8
Process  Burst Time     Waiting Time    Turnaround Time
p1           6               0               6
p4           8               6               14
p3           9               14              23
p2           9               23              32

Average Waiting Time=10.750000
Average Turnaround Time=18.750000
```

EXPERIMENT-3

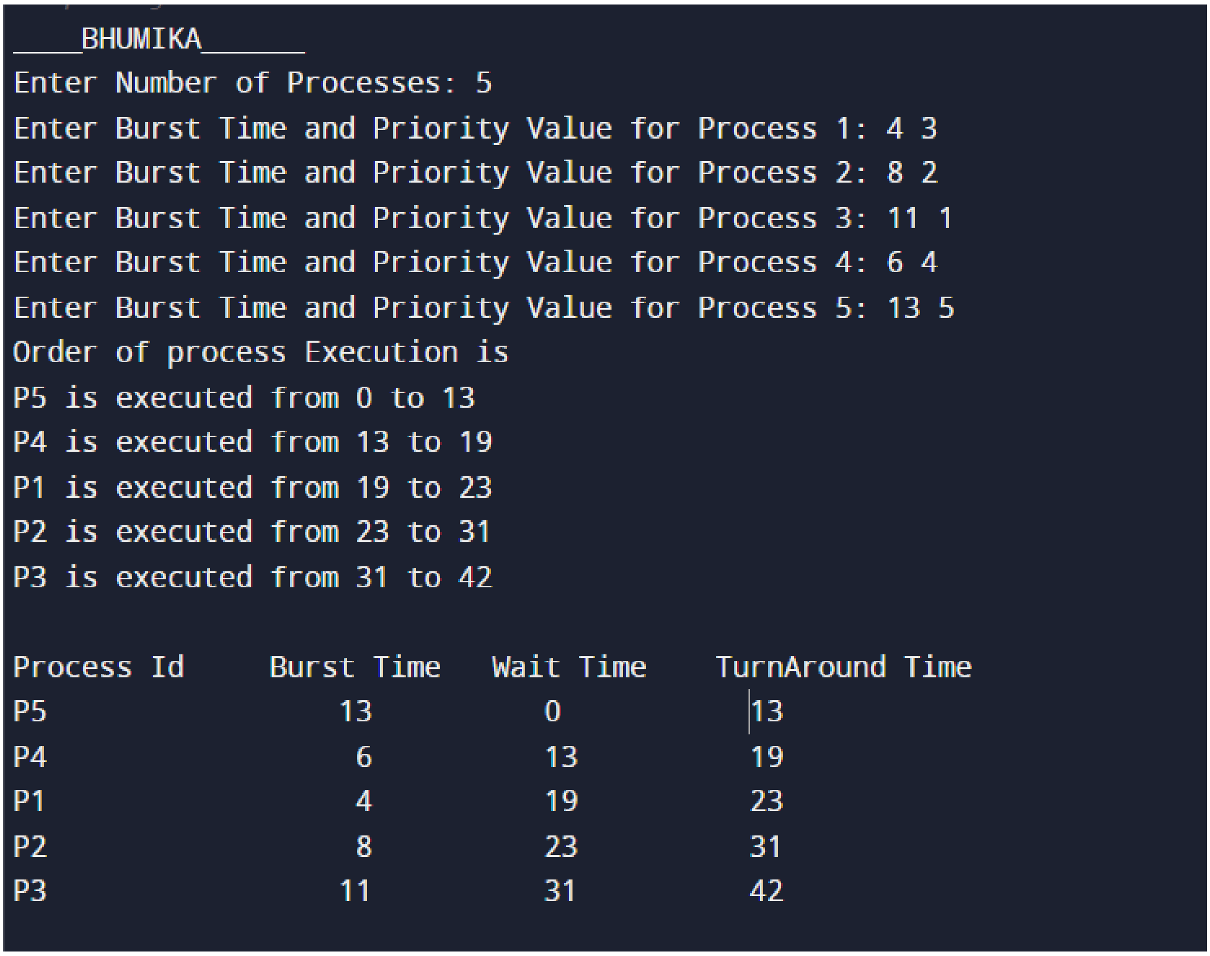
AIM: Write a program to perform priority scheduling.

PROGRAM CODE:

```
#include <stdio.h>
void swap(int *a,int *b)
{
   int temp=*a;
   *a=*b;
   *b=temp;
}
int main()
{
   int n;
   printf("____BHUMIKA______\n");
   printf("Enter Number of Processes: ");
   scanf("%d",&n);
   // b is array for burst time, p for priority and index for process id
   int b[n],p[n],index[n];
   for(int i=0;i<n;i++)
   {
      printf("Enter Burst Time and Priority Value for Process %d: ",i+1);
      scanf("%d %d",&b[i],&p[i]);
      index[i]=i+1;
}
for(int i=0;i<n;i++)
{
    int a=p[i],m=i;

    //Finding out highest priority element and placing it at its desired position
    for(int j=i;j<n;j++)
    {
        if(p[j] > a)
        {
            a=p[j];
            m=j;
        }
    }

    //Swapping processes
    swap(&p[i], &p[m]);
    swap(&b[i], &b[m]);
    swap(&index[i],&index[m]);
}

// T stores the starting time of process
int t=0;
```

```
   //Printing scheduled process
   printf("Order of process Execution is\n");
   for(int i=0;i<n;i++)
   {
      printf("P%d is executed from %d to %d\n",index[i],t,t+b[i]);
      t+=b[i];
   }
   printf("\n");
   printf("Process Id     Burst Time   Wait Time    TurnAround Time\n");
   int wait_time=0;
   for(int i=0;i<n;i++)
   {
      printf("P%d         %d         %d         %d\n",index[i],b[i],wait_time,wait_time
+ b[i]);
      wait_time += b[i];
   }
   return 0;
}
```

OUTPUT:



```
____BHUMIKA______
Enter Number of Processes: 5
Enter Burst Time and Priority Value for Process 1: 4 3
Enter Burst Time and Priority Value for Process 2: 8 2
Enter Burst Time and Priority Value for Process 3: 11 1
Enter Burst Time and Priority Value for Process 4: 6 4
Enter Burst Time and Priority Value for Process 5: 13 5
Order of process Execution is
P5 is executed from 0 to 13
P4 is executed from 13 to 19
P1 is executed from 19 to 23
P2 is executed from 23 to 31
P3 is executed from 31 to 42

Process Id     Burst Time    Wait Time     TurnAround Time
P5                 13           0              13
P4                  6          13              19
P1                  4          19              23
P2                  8          23              31
P3                 11          31              42
```

EXPERIMENT-4

AIM: Write a program to implement CPU scheduling for Round Robin.

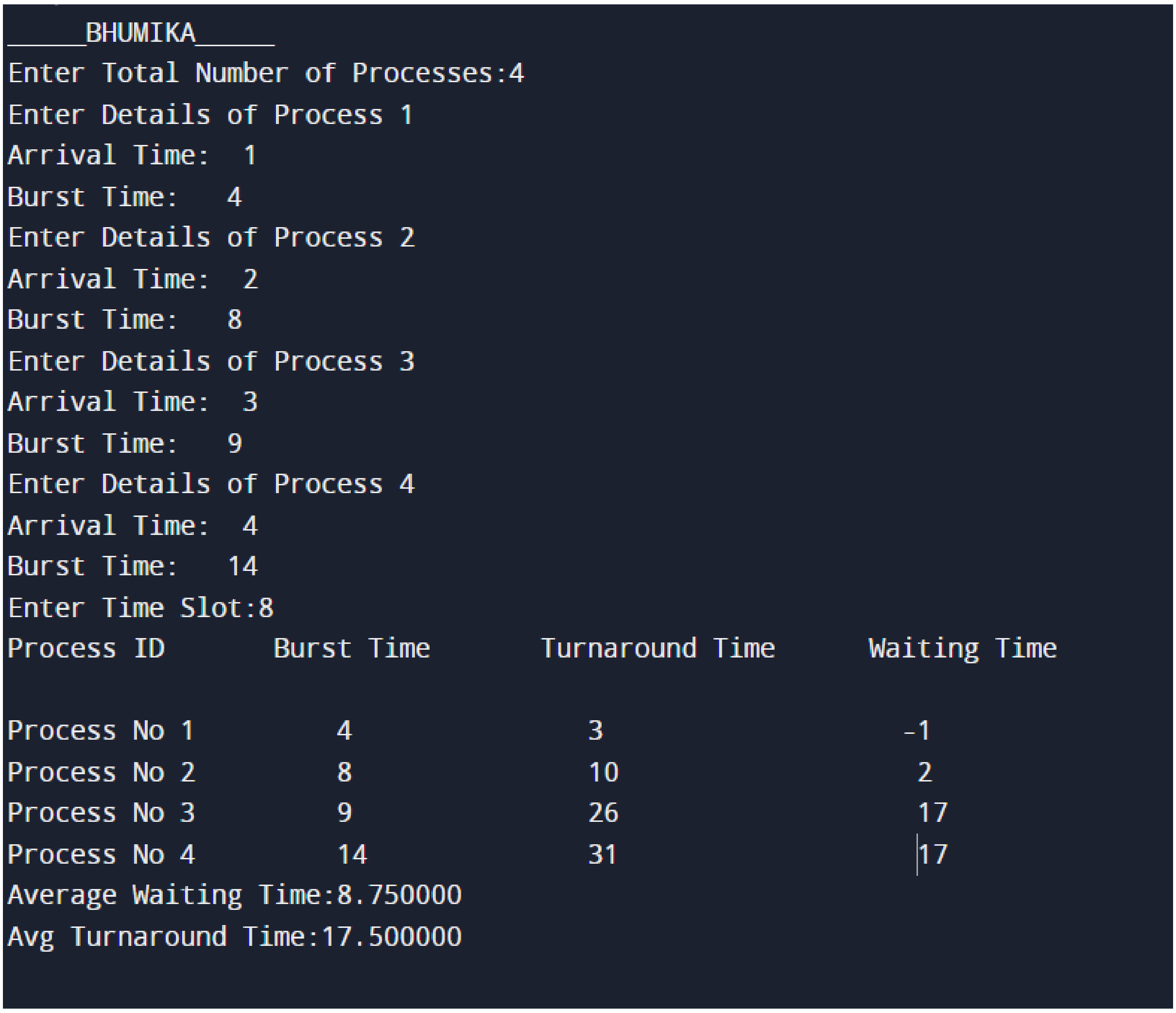
PROGRAM CODE:

```
#include<stdio.h>

int main()
{
   int  n;
   printf("_____BHUMIKA_____\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;
   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];
   }
```

```
int time_slot;
printf("Enter Time Slot:");
scanf("%d", &time_slot);

int total = 0,  counter = 0,i;
printf("Process ID      Burst Time      Turnaround Time      Waiting Time\n");
for(total=0, i = 0; x!=0; )
{
   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--;
      printf("\nProcess No %d  \t\t %d\t\t\t\t %d\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("\nAvg Turnaround Time:%f", average_turnaround_time);
return 0;
```

```
}
```

OUTPUT:



```
_____BHUMIKA_____
Enter Total Number of Processes:4
Enter Details of Process 1
Arrival Time:  1
Burst Time:   4
Enter Details of Process 2
Arrival Time:  2
Burst Time:   8
Enter Details of Process 3
Arrival Time:  3
Burst Time:   9
Enter Details of Process 4
Arrival Time:  4
Burst Time:   14
Enter Time Slot:8
Process ID       Burst Time       Turnaround Time      Waiting Time

Process No 1         4               3                    -1
Process No 2         8               10                    2
Process No 3         9               26                    17
Process No 4         14              31                    17
Average Waiting Time:8.750000
Avg Turnaround Time:17.500000
```

EXPERIMENT-5

AIM: Write a program for page replacement policy using

a. **LRU**

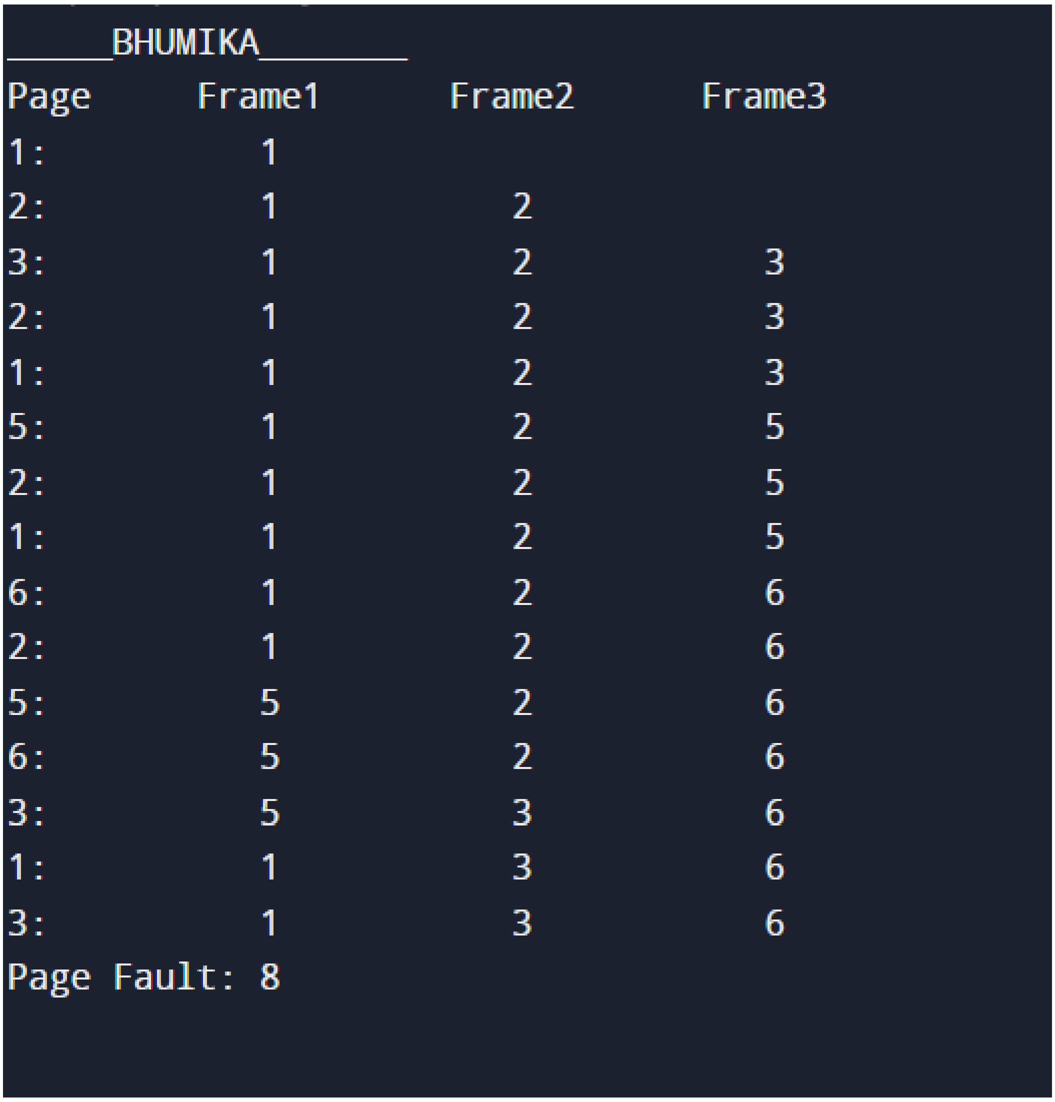
PROGRAM CODE:

```
#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("_____BHUMIKA_______\n");
    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;

                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:



```
______BHUMIKA________
Page       Frame1         Frame2         Frame3
1:             1
2:             1              2
3:             1              2              3
2:             1              2              3
1:             1              2              3
5:             1              2              5
2:             1              2              5
1:             1              2              5
6:             1              2              6
2:             1              2              6
5:             5              2              6
6:             5              2              6
3:             5              3              6
1:             1              3              6
3:             1              3              6
Page Fault: 8
```

b. FIFO:

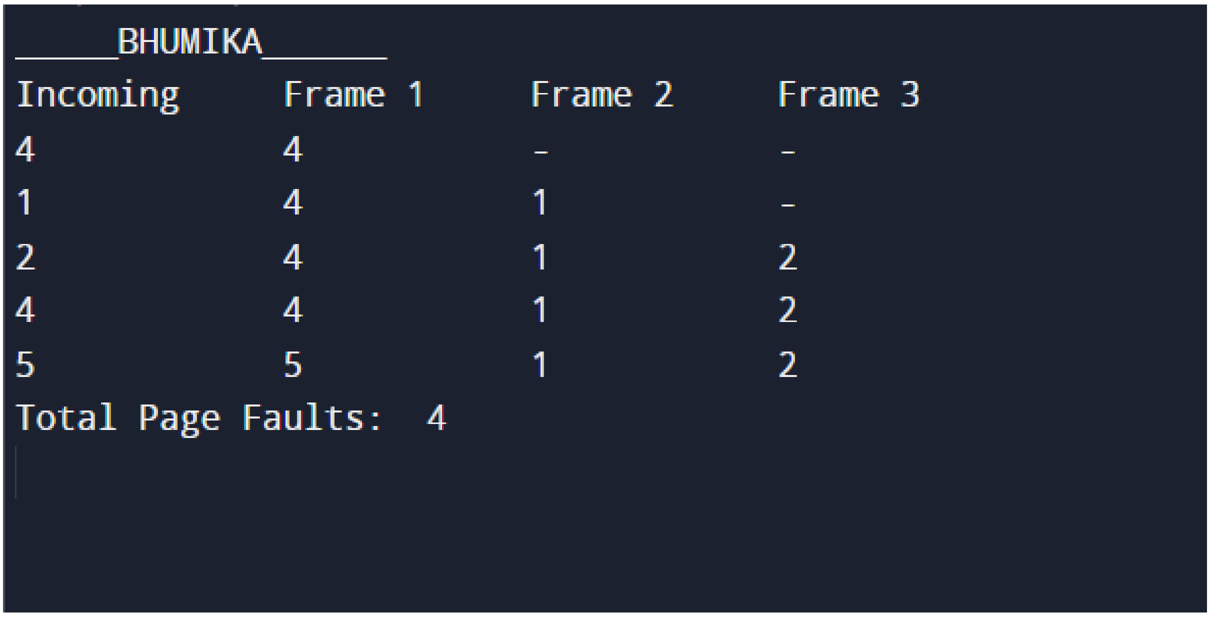
PROGRAM CODE:

```
#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("_____BHUMIKA______\n");

    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:



```
_____BHUMIKA______
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
```

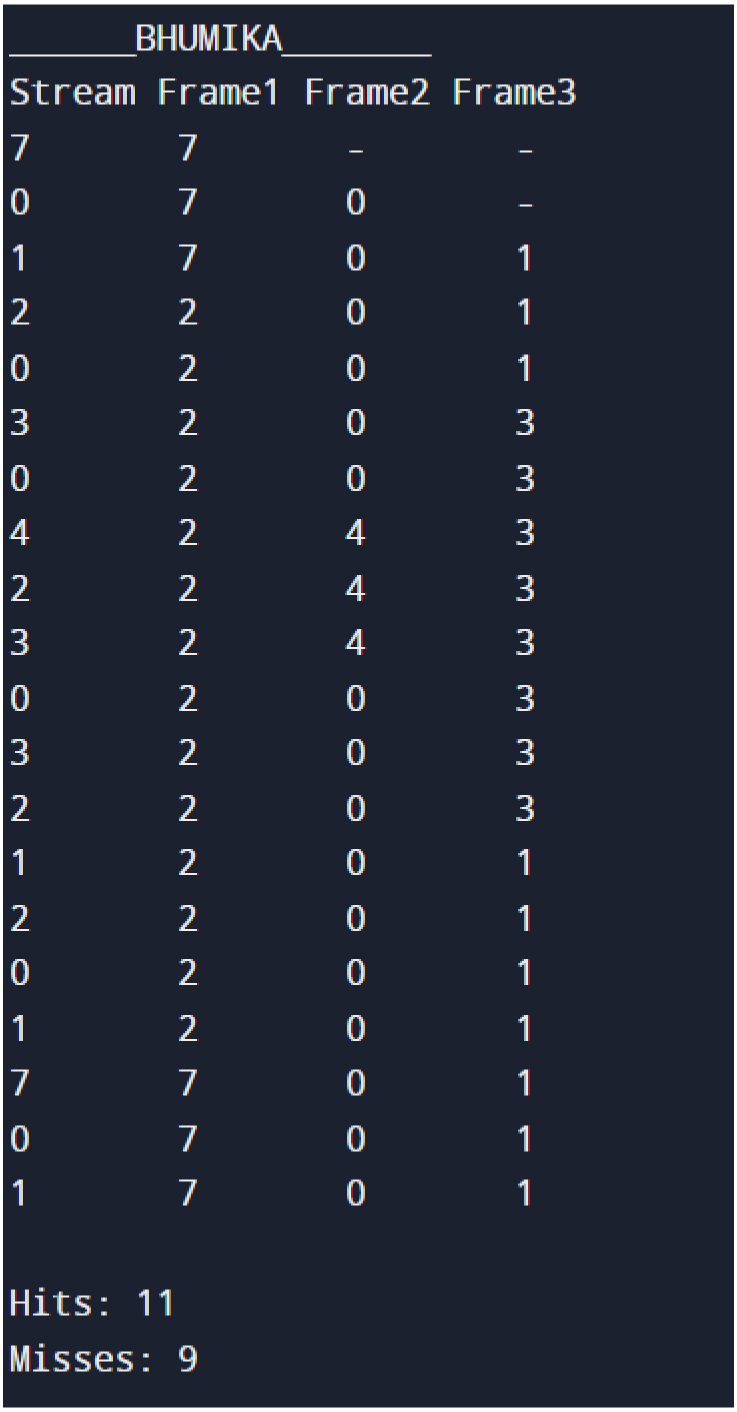
c. Optimal:

PROGRAM 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("______BHUMIKA_______\n");
   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[] = {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:



```
______BHUMIKA_______
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
```

EXPERIMENT-6

AIM: Write a program to implement first fit,best fit and worst fit algorithm for memory management.

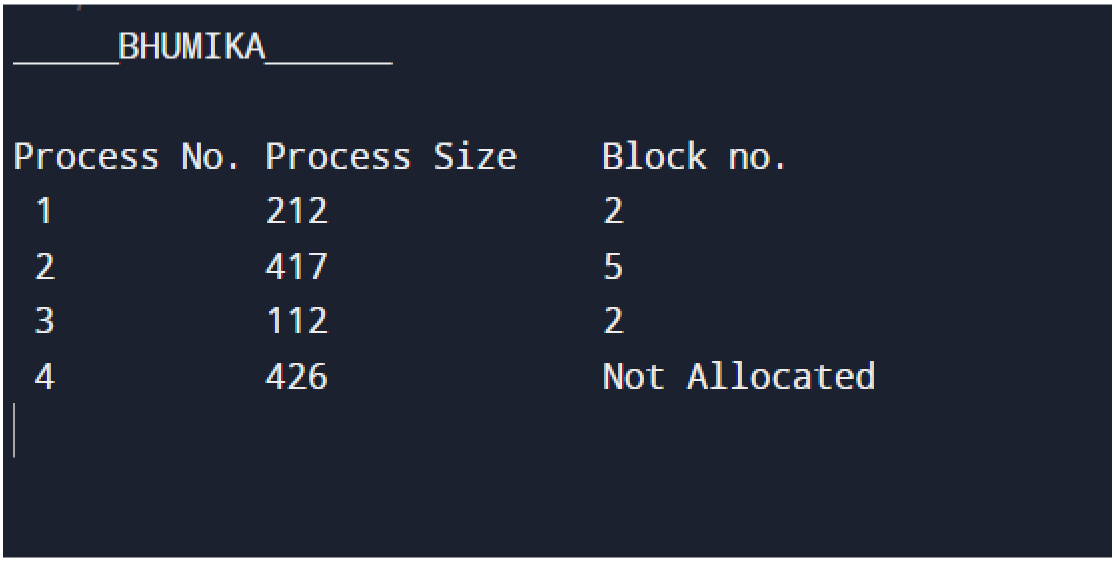
- **First Fit:**

PROGRAM CODE:

```
#include<stdio.h>
void firstFit(int blockSize[], int m, int processSize[], int n)
{
        int i, j;
        int allocation[n];
        for(i = 0; i < n; i++)
        {
                allocation[i] = -1;
        }
        for (i = 0; i < n; i++)
        {
                for (j = 0; j < m; j++)
                {
                        if (blockSize[j] >= processSize[i])
                        {
                                allocation[i] = j;
                                blockSize[j] -= processSize[i];
                break;
            }
        }
    }
  printf("_____BHUMIKA______\n");
    printf("\nProcess No.\tProcess Size\tBlock no.\n");
    for (int i = 0; i < n; i++)
    {
        printf(" %i\t\t\t", i+1);
        printf("%i\t\t\t\t", processSize[i]);
        if (allocation[i] != -1)
            printf("%i", allocation[i] + 1);
        else
            printf("Not Allocated");
        printf("\n");
    }
}
int main()
{
    int m;
    int n;
    int blockSize[] = {100, 500, 200, 300, 600};
    int processSize[] = {212, 417, 112, 426};
```

```
        m = sizeof(blockSize) / sizeof(blockSize[0]);

        n = sizeof(processSize) / sizeof(processSize[0]);

        firstFit(blockSize, m, processSize, n);

        return 0 ;

}
```

OUTPUT:



```
_____BHUMIKA______

Process No. Process Size    Block no.
 1          212             2
 2          417             5
 3          112             2
 4          426             Not Allocated
```

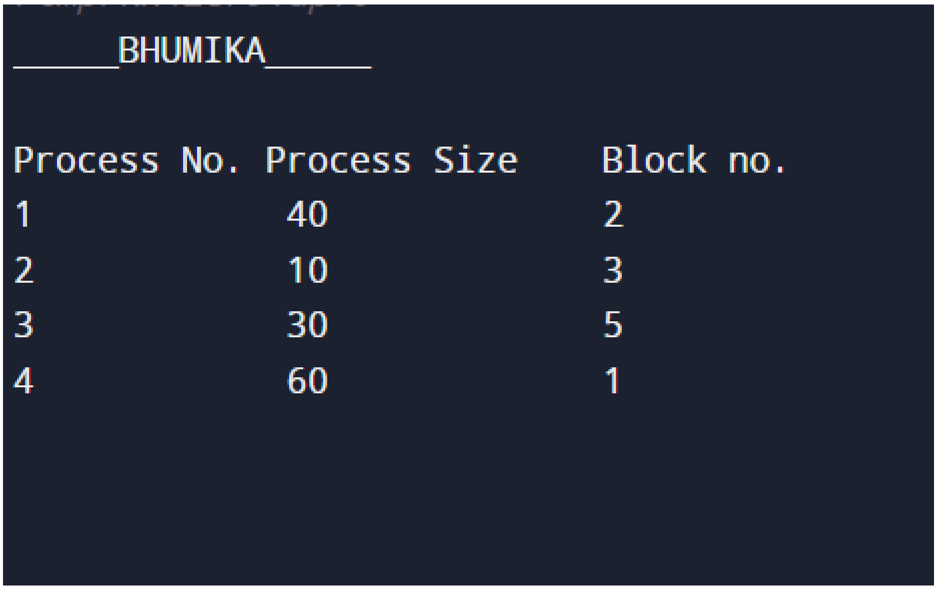
- **Best Fit:**

PROGRAM CODE:

```
#include <stdio.h>

void implimentBestFit(int blockSize[], int blocks, int processSize[], int proccesses)

{

    int allocation[proccesses];

    int occupied[blocks];

    for(int i = 0; i < proccesses; i++){

        allocation[i] = -1;

    }


    for(int i = 0; i < blocks; i++){

        occupied[i] = 0;

    }

    for (int i = 0; i < proccesses; i++)

    {


        int indexPlaced = -1;

        for (int j = 0; j < blocks; j++) {

            if (blockSize[j] >= processSize[i] && !occupied[j])

            {

                if (indexPlaced == -1)
                indexPlaced = j;
             else if (blockSize[j] < blockSize[indexPlaced])
                indexPlaced = j;
          }
       }
       if (indexPlaced != -1)
       {
          allocation[i] = indexPlaced;
          occupied[indexPlaced] = 1;
       }
    }
    printf("_____BHUMIKA_____\n");
    printf("\nProcess No.\tProcess Size\tBlock no.\n");
    for (int i = 0; i < proccesses; i++)
    {
       printf("%d \t\t\t %d \t\t\t", i+1, processSize[i]);
       if (allocation[i] != -1)
          printf("%d\n",allocation[i] + 1);
       else
          printf("Not Allocated\n");
    }
}
```

```
int main()
{
   int blockSize[] = {100, 50, 30, 120, 35};
   int processSize[] = {40, 10, 30, 60};
   int blocks = sizeof(blockSize)/sizeof(blockSize[0]);
   int proccesses = sizeof(processSize)/sizeof(processSize[0]);

   implimentBestFit(blockSize, blocks, processSize, proccesses);

   return 0 ;
}
```

OUTPUT:



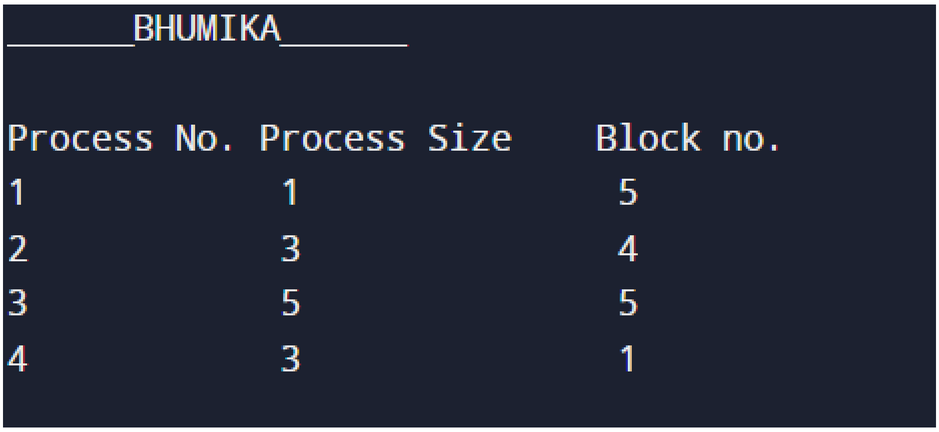
```
_____BHUMIKA_____

Process No. Process Size    Block no.
1             40             2
2             10             3
3             30             5
4             60             1
```

- **Worst Fit:**

PROGRAM CODE:

```
#include <stdio.h>

void implimentWorstFit(int blockSize[], int blocks, int processSize[], int
processes)

{

    int allocation[processes];

    for(int i = 0; i < processes; i++){

        allocation[i] = -1;

    }

    for (int i=0; i<processes; i++)

    {


        int indexPlaced = -1;

        for (int j=0; j<blocks; j++)

        {

            if (blockSize[j] >= processSize[i])

            {

                if (indexPlaced == -1)

                    indexPlaced = j;

                else if (blockSize[indexPlaced] < blockSize[j])

                    indexPlaced = j;

            }
        }
        if (indexPlaced != -1)
        {
            allocation[i] = indexPlaced;
            blockSize[indexPlaced] -= processSize[i];
        }
    }
    printf("______BHUMIKA______\n");
    printf("\nProcess No.\tProcess Size\tBlock no.\n");
    for (int i = 0; i < processes; i++)
    {
        printf("%d \t\t\t %d \t\t\t", i+1, processSize[i]);
        if (allocation[i] != -1)
            printf("%d\n",allocation[i] + 1);
        else
            printf("Not Allocated\n");
    }
}
int main()
{
    int blockSize[] = {5, 4, 3, 6, 7};
    int processSize[] = {1, 3, 5, 3};
    int blocks = sizeof(blockSize)/sizeof(blockSize[0]);
    int processes = sizeof(processSize)/sizeof(processSize[0]);


    implimentWorstFit(blockSize, blocks, processSize, processes);


    return 0 ;
}
```

OUTPUT:



```
______BHUMIKA______

Process No. Process Size    Block no.
1            1               5
2            3               4
3            5               5
4            3               1
```

EXPERIMENT-7

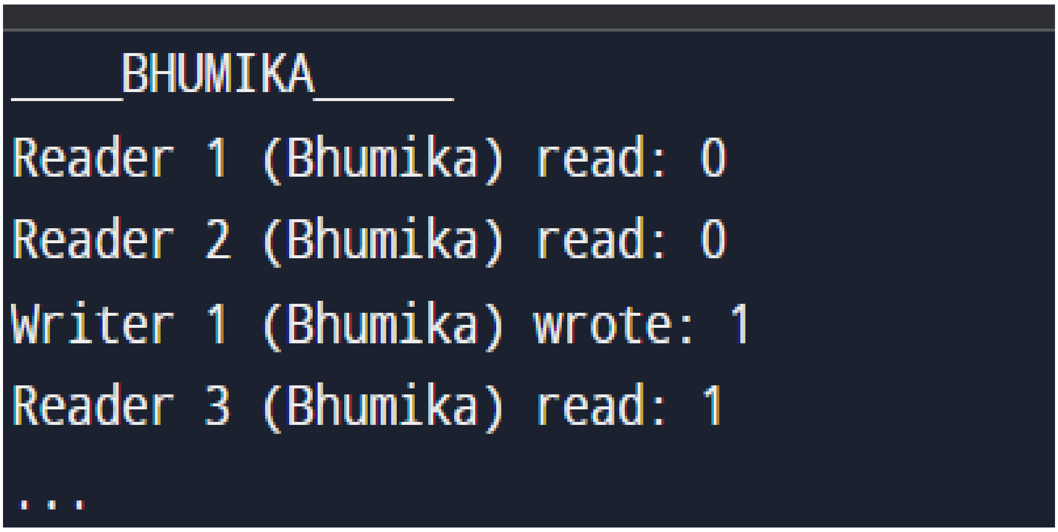
AIM: Write a program to implement reader/writer problem using semaphore.

PROGRAM CODE:

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

#define NUM_READERS 3
#define NUM_WRITERS 2

sem_t mutex, writeBlock;
int readersCount = 0;
int sharedData = 0;

void *reader(void *arg) {
    int readerId = *(int *)arg;
    while (1) {
        sem_wait(&mutex);
        readersCount++;
        if (readersCount == 1) {
            sem_wait(&writeBlock);
        }
        sem_post(&mutex);

        // Read the shared data
        printf("Reader %d (Bhumika) read: %d\n", readerId, sharedData);

        sem_wait(&mutex);
        readersCount--;
        if (readersCount == 0) {
            sem_post(&writeBlock);
        }
        sem_post(&mutex);

        // Sleep to simulate processing
        usleep(100000);
    }
    return NULL;
}

void *writer(void *arg) {
    int writerId = *(int *)arg;
    while (1) {
        sem_wait(&writeBlock);
        // Write to the shared data
        sharedData++;
        printf("Writer %d (Bhumika) wrote: %d\n", writerId, sharedData);

        sem_post(&writeBlock);

        // Sleep to simulate processing
        usleep(200000);
    }
    return NULL;
}

int main() {
    pthread_t readers[NUM_READERS];
    pthread_t writers[NUM_WRITERS];

    sem_init(&mutex, 0, 1);
    sem_init(&writeBlock, 0, 1);

    int readerIds[NUM_READERS];
    int writerIds[NUM_WRITERS];

    for (int i = 0; i < NUM_READERS; i++) {
}
```

```
        readerIds[i] = i + 1;
        pthread_create(&readers[i], NULL, reader, &readerIds[i]);
    }

    for (int i = 0; i < NUM_WRITERS; i++) {
        writerIds[i] = i + 1;
        pthread_create(&writers[i], NULL, writer, &writerIds[i]);
    }

    for (int i = 0; i < NUM_READERS; i++) {
        pthread_join(readers[i], NULL);
    }

    for (int i = 0; i < NUM_WRITERS; i++) {
        pthread_join(writers[i], NULL);
    }

    sem_destroy(&mutex);
    sem_destroy(&writeBlock);

    return 0;
}
```

OUTPUT:



```
____BHUMIKA_____
Reader 1 (Bhumika) read: 0
Reader 2 (Bhumika) read: 0
Writer 1 (Bhumika) wrote: 1
Reader 3 (Bhumika) read: 1
...
```

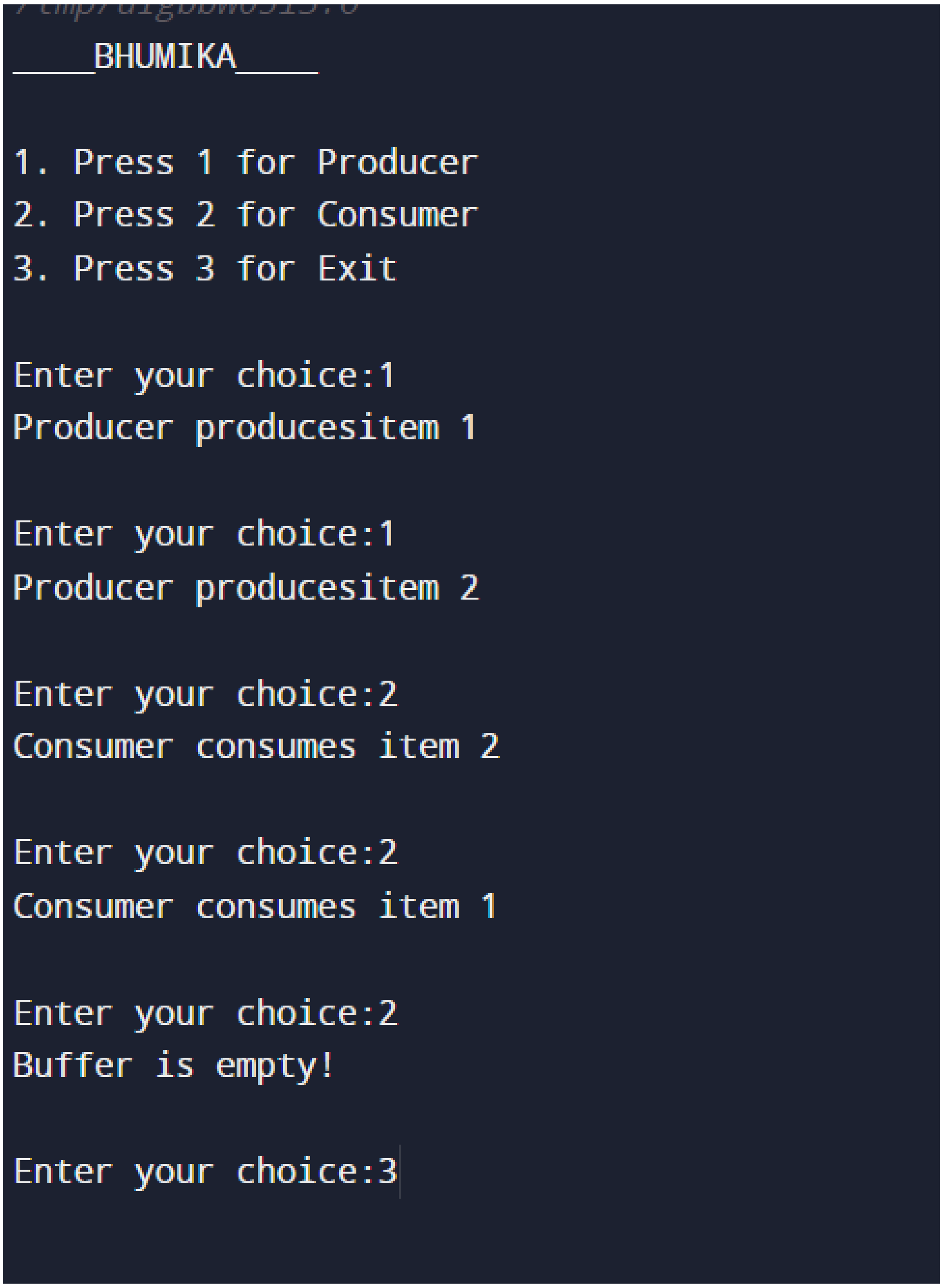
EXPERIMENT-8

AIM: Write a program to implement Producer-Consumer problem using semaphores.

PROGRAM CODE:

```
#include <stdio.h>

#include <stdlib.h>


int mutex = 1;


int full = 0;


int empty = 10, x = 0;


void producer()

{

        --mutex;


        ++full;

        --empty;

        x++;


        printf("\nProducer produces"

                "item %d",
                x);

        ++mutex;
}

void consumer()
{
        --mutex;
        --full;
        ++empty;
        printf("\nConsumer consumes "
                "item %d",
                x);
        x--;

        ++mutex;
}

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

#pragma omp critical

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

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

                // Switch Cases
                switch (n) {
                case 1:
                    if ((mutex == 1)
                        && (empty != 0)) {
                        producer();
                    }
                    else {
                        printf("Buffer is full!");
                    }
                    break;

                case 2:
                    if ((mutex == 1)
                        && (full != 0)) {
                        consumer();
                    }

                    else {
                        printf("Buffer is empty!");
                    }
                    break;
                case 3:
                    exit(0);
                    break;
                }
        }
}
```

OUTPUT:



```
/tmp/uIgbbWo515.o
____BHUMIKA____

1. Press 1 for Producer
2. Press 2 for Consumer
3. Press 3 for Exit

Enter your choice:1
Producer producesitem 1

Enter your choice:1
Producer producesitem 2

Enter your choice:2
Consumer consumes item 2

Enter your choice:2
Consumer consumes item 1

Enter your choice:2
Buffer is empty!

Enter your choice:3
```

EXPERIMENT-9

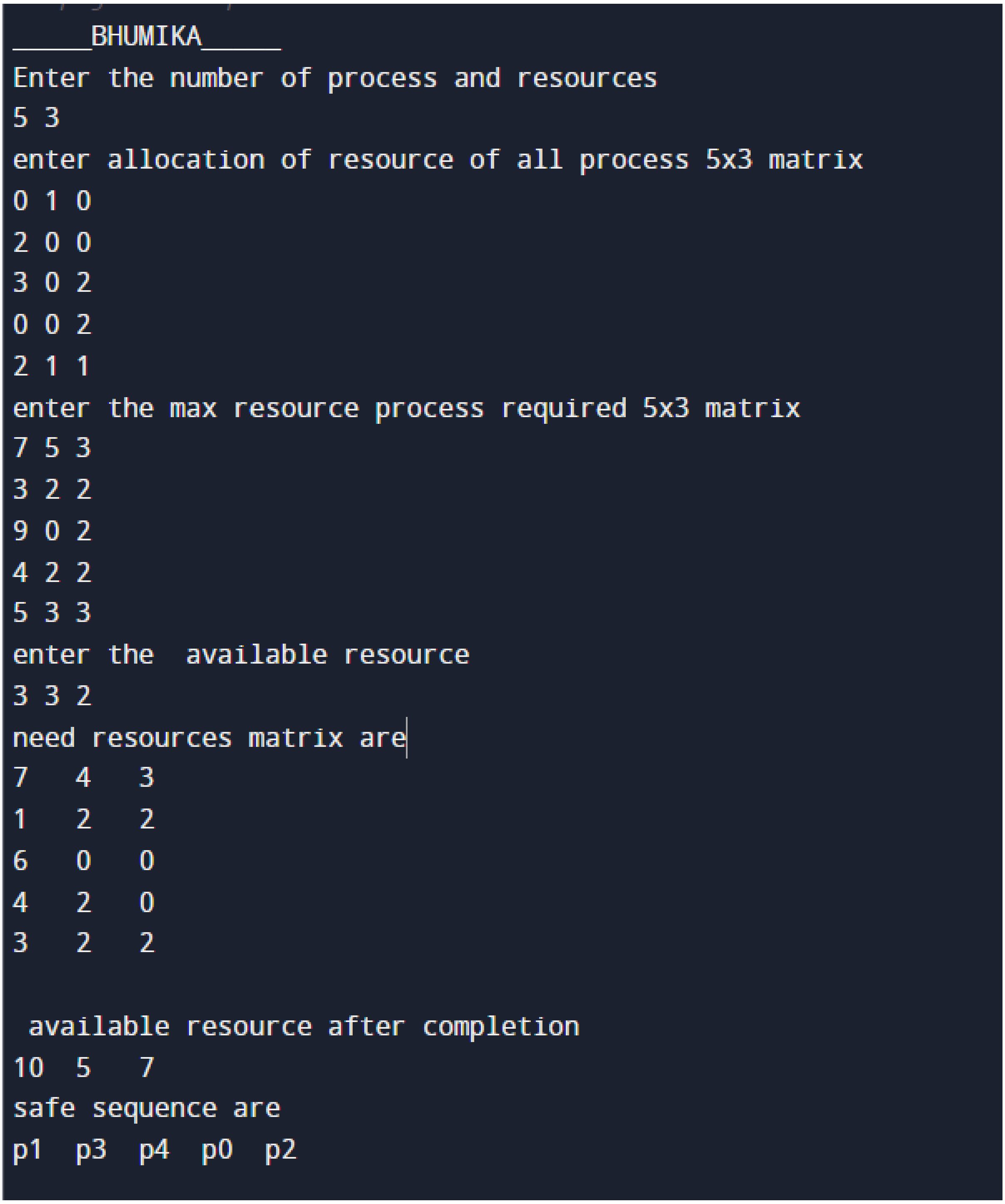
AIM: Write a program to implement Banker’s algorithm for deadlock avoidance.

PROGRAM CODE:

```
#include<stdio.h>


int main() {
  int p, c, count = 0, i, j, alc[5][3], max[5][3], need[5][3], safe[5], available[3], done[5], terminate = 0;
  printf("_____BHUMIKA_____\n");
  printf("Enter the number of process and resources");
  scanf("%d %d", & p, & c);


  printf("enter allocation of resource of all process %dx%d matrix", p, c);
  for (i = 0; i < p; i++) {
    for (j = 0; j < c; j++) {
      scanf("%d", & alc[i][j]);
    }
  }
  printf("enter the max resource process required %dx%d matrix", p, c);
  for (i = 0; i < p; i++) {
    for (j = 0; j < c; j++) {
      scanf("%d", & max[i][j]);
  }
}
printf("enter the  available resource");
for (i = 0; i < c; i++)
  scanf("%d", & available[i]);
printf("\n need resources matrix are\n");
for (i = 0; i < p; i++) {
  for (j = 0; j < c; j++) {
    need[i][j] = max[i][j] - alc[i][j];
    printf("%d\t", need[i][j]);
  }
  printf("\n");
}
for (i = 0; i < p; i++) {
  done[i] = 0;
}
while (count < p) {
  for (i = 0; i < p; i++) {
    if (done[i] == 0) {
      for (j = 0; j < c; j++) {
        if (need[i][j] > available[j])
          break;
      }
```

```
   if (j == c) {
    safe[count] = i;
    done[i] = 1;

    for (j = 0; j < c; j++) {
     available[j] += alc[i][j];
    }
    count++;
    terminate = 0;
   } else {
    terminate++;
   }
  }
 }
 if (terminate == (p - 1)) {
  printf("safe sequence does not exist");
  break;
 }

}
if (terminate != (p - 1)) {
 printf("\n available resource after completion\n");
 for (i = 0; i < c; i++) {
      printf("%d\t", available[i]);
    }
    printf("\n safe sequence are\n");
    for (i = 0; i < p; i++) {
      printf("p%d\t", safe[i]);
    }
  }

  return 0;
}
```

OUTPUT:



```
_____BHUMIKA_____
Enter the number of process and resources
5 3
enter allocation of resource of all process 5x3 matrix
0 1 0
2 0 0
3 0 2
0 0 2
2 1 1
enter the max resource process required 5x3 matrix
7 5 3
3 2 2
9 0 2
4 2 2
5 3 3
enter the  available resource
3 3 2
need resources matrix are
7   4   3
1   2   2
6   0   0
4   2   0
3   2   2

 available resource after completion
10  5   7
safe sequence are
p1  p3  p4  p0  p2
```

EXPERIMENT-10

AIM: Write C programs to implement the various File Organisation Techniques.

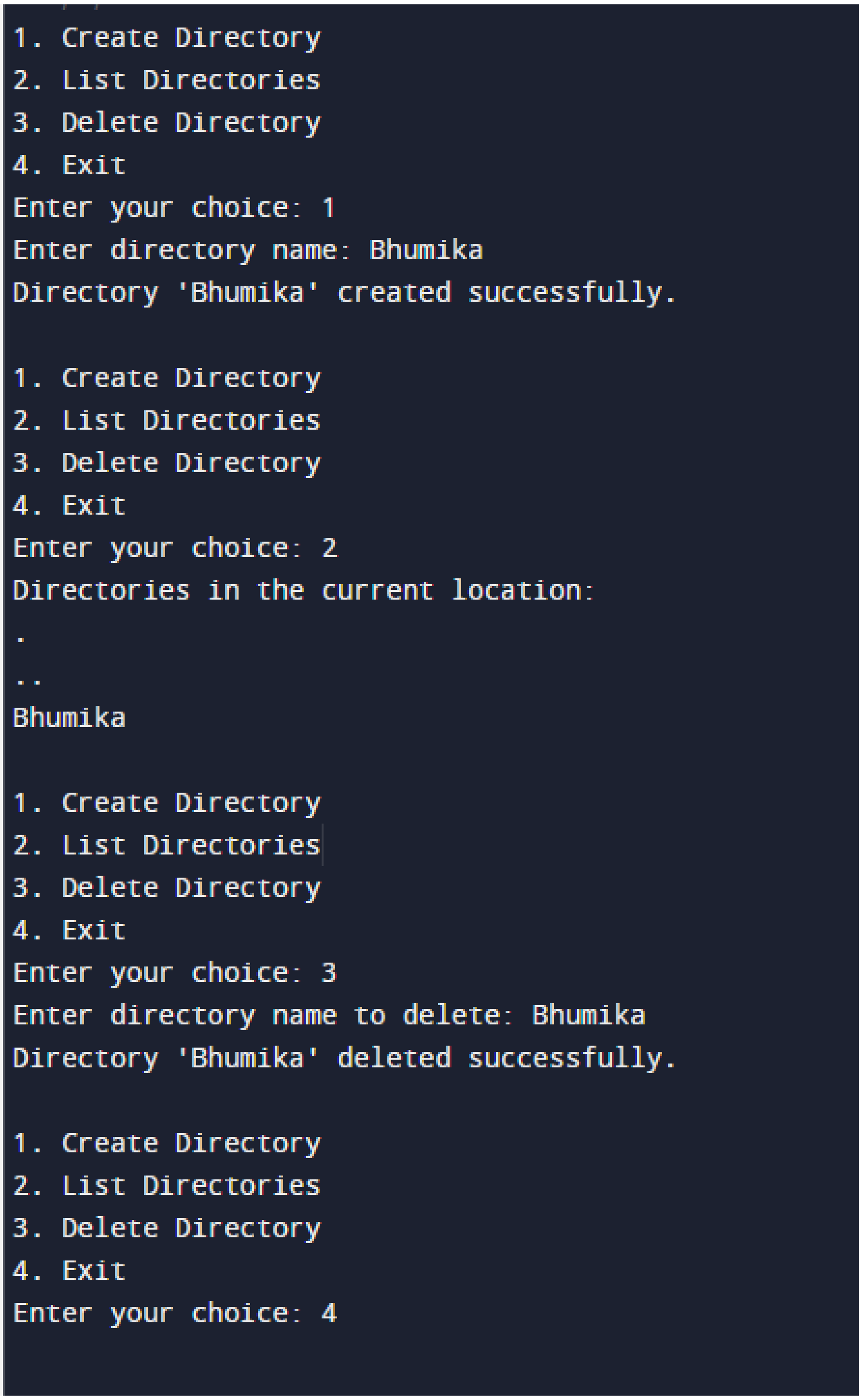
PROGRAM CODE:

- **SINGLE LEVEL DIRECTORY:**

```
#include <stdio.h>

#include <stdlib.h>

#include <sys/stat.h>

#include <unistd.h>

#include <dirent.h>

#include <string.h>


void createDirectory(const char *directoryName) {

    int status = mkdir(directoryName, 0777);


    if (status == 0) {

        printf("Directory '%s' created successfully.\n", directoryName);

    } else {

        printf("Failed to create directory '%s'.\n", directoryName);

    }

}


void listDirectories() {

    DIR *dir;
    struct dirent *entry;

    if ((dir = opendir(".")) != NULL) {
        printf("Directories in the current location:\n");
        while ((entry = readdir(dir)) != NULL) {
            if (entry->d_type == DT_DIR) {
                printf("%s\n", entry->d_name);
            }
        }
        closedir(dir);
    } else {
        printf("Error opening directory.\n");
    }
}

void deleteDirectory(const char *directoryName) {
    if (rmdir(directoryName) == 0) {
        printf("Directory '%s' deleted successfully.\n", directoryName);
    } else {
        printf("Failed to delete directory '%s'.\n", directoryName);
    }
}
```

```
int main() {
    int choice;
    char dirName[100];

    while (1) {
        printf("\n1. Create Directory\n");
        printf("2. List Directories\n");
        printf("3. Delete Directory\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter directory name: ");
                scanf("%s", dirName);
                createDirectory(dirName);
                break;
            case 2:
                listDirectories();
                break;
            case 3:
                printf("Enter directory name to delete: ");
                scanf("%s", dirName);

                deleteDirectory(dirName);

                break;

            case 4:

                exit(0);

            default:

                printf("Invalid choice. Please enter a valid option.\n");

        }

    }


    return 0;

}
```

OUTPUT:



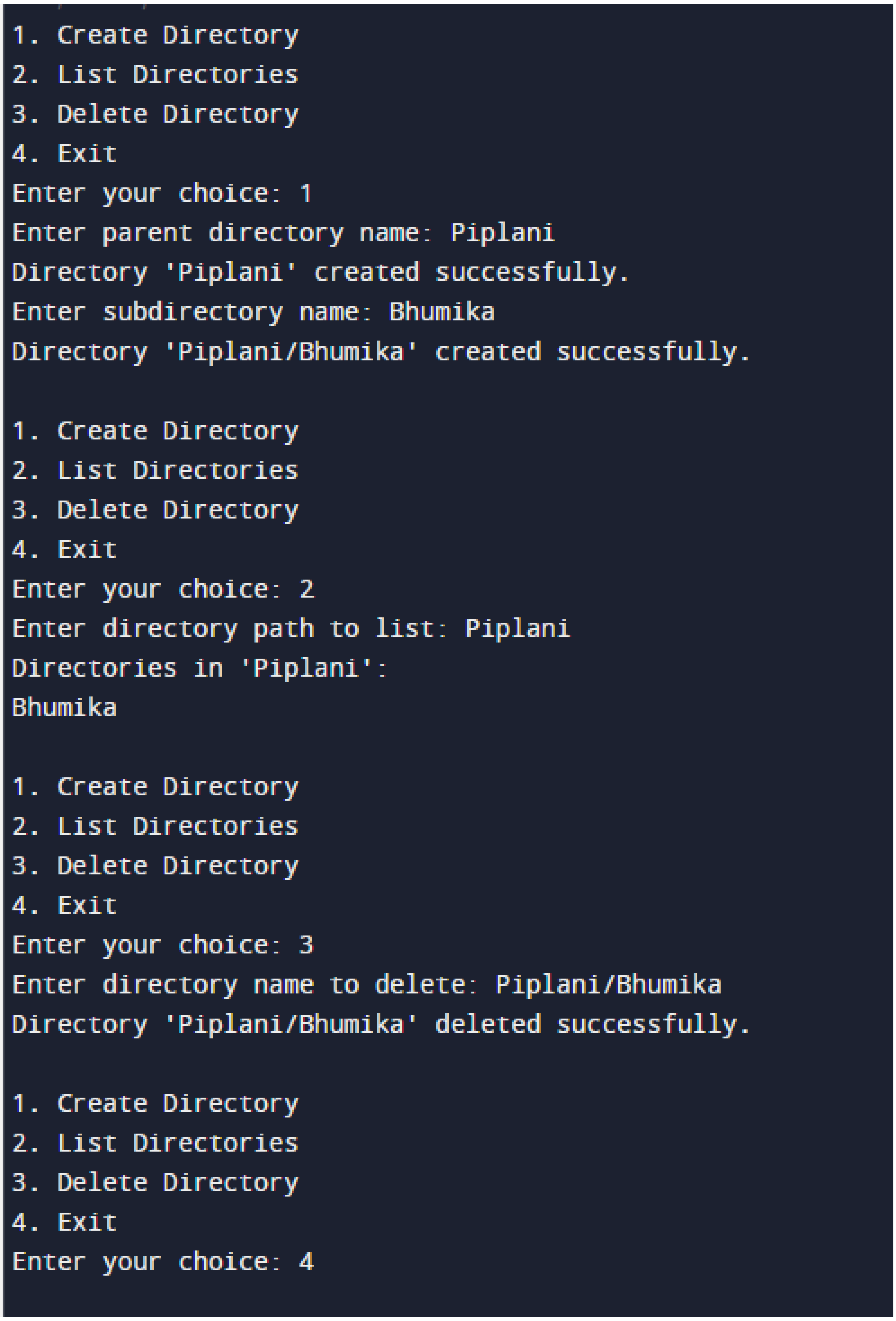
```
1. Create Directory
2. List Directories
3. Delete Directory
4. Exit
Enter your choice: 1
Enter directory name: Bhumika
Directory 'Bhumika' created successfully.

1. Create Directory
2. List Directories
3. Delete Directory
4. Exit
Enter your choice: 2
Directories in the current location:
.
..
Bhumika

1. Create Directory
2. List Directories
3. Delete Directory
4. Exit
Enter your choice: 3
Enter directory name to delete: Bhumika
Directory 'Bhumika' deleted successfully.

1. Create Directory
2. List Directories
3. Delete Directory
4. Exit
Enter your choice: 4
```

➢ **Two Level Directory:**

```
#include <stdio.h>

#include <stdlib.h>

#include <sys/stat.h>

#include <unistd.h>

#include <dirent.h>

#include <string.h>


void createDirectory(const char *directoryName) {

   int status = mkdir(directoryName, 0777);


   if (status == 0) {

      printf("Directory '%s' created successfully.\n", directoryName);

   } else {

      printf("Failed to create directory '%s'.\n", directoryName);

   }

}


void listDirectories(const char *path) {

   DIR *dir;

   struct dirent *entry;


   if ((dir = opendir(path)) != NULL) {
        printf("Directories in '%s':\n", path);
        while ((entry = readdir(dir)) != NULL) {
            if (entry->d_type == DT_DIR && strcmp(entry->d_name, ".") != 0 && strcmp(entry->d_name, "..") != 0) {
                printf("%s\n", entry->d_name);
            }
        }
        closedir(dir);
    } else {
        printf("Error opening directory '%s'.\n", path);
    }
}

void deleteDirectory(const char *directoryName) {
    if (rmdir(directoryName) == 0) {
        printf("Directory '%s' deleted successfully.\n", directoryName);
    } else {
        printf("Failed to delete directory '%s'.\n", directoryName);
    }
}
  Int main() {
    int choice;
    char dirName[100];
    char subDirName[100];
    char path[200];

    while (1) {
        printf("\n1. Create Directory\n");
        printf("2. List Directories\n");
        printf("3. Delete Directory\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter parent directory name: ");
                scanf("%s", dirName);
                createDirectory(dirName);

                printf("Enter subdirectory name: ");
                scanf("%s", subDirName);

                snprintf(path, sizeof(path), "%s/%s", dirName, subDirName);
                createDirectory(path);
                break;
```

```
          case 2:
             printf("Enter directory path to list: ");
             scanf("%s", path);
             listDirectories(path);
             break;
          case 3:
             printf("Enter directory name to delete: ");
             scanf("%s", dirName);
             deleteDirectory(dirName);
             break;
          case 4:
             exit(0);
          default:
             printf("Invalid choice. Please enter a valid option.\n");
       }
    }

    return 0;
}
```

OUTPUT:



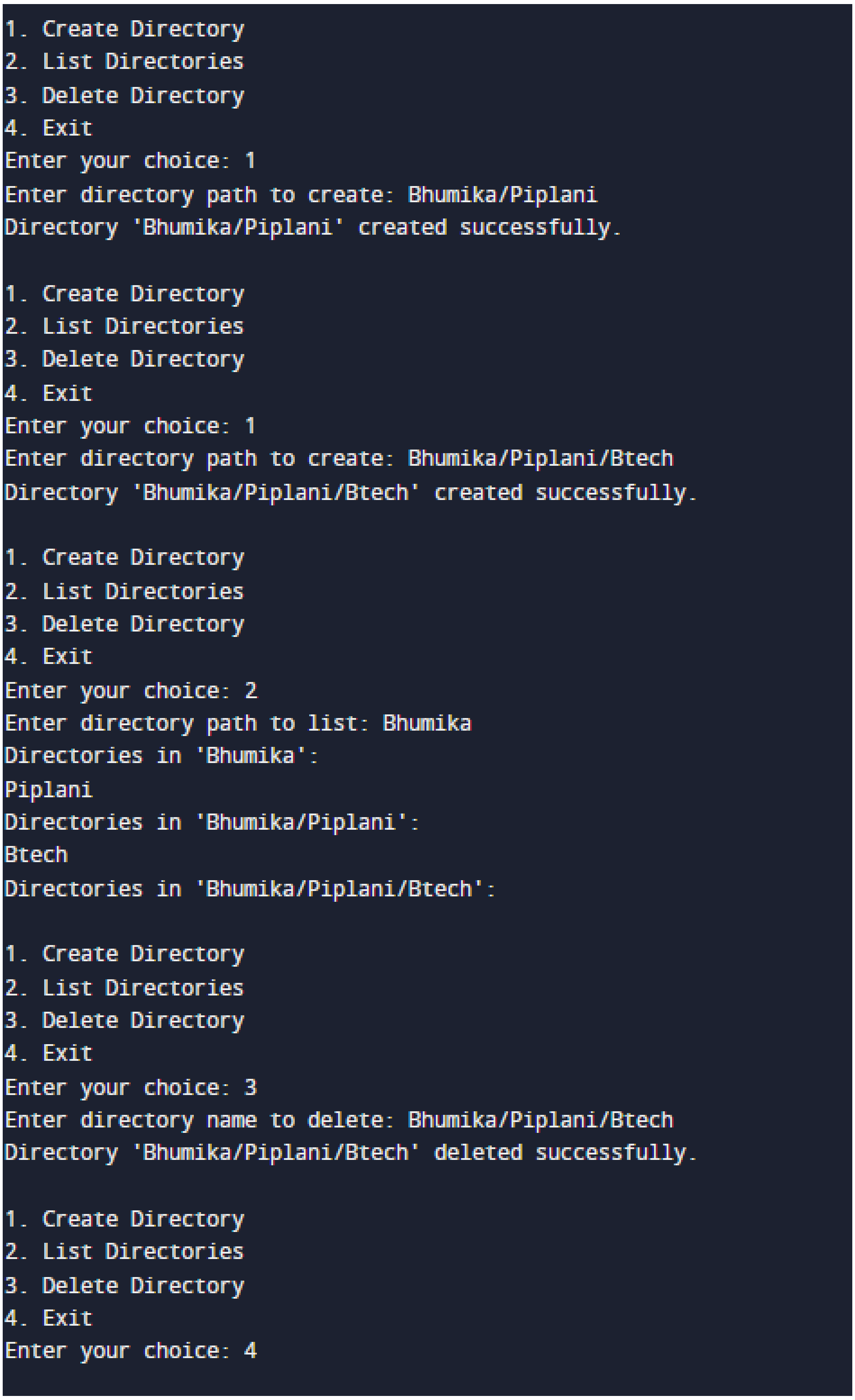
```
1. Create Directory
2. List Directories
3. Delete Directory
4. Exit
Enter your choice: 1
Enter parent directory name: Piplani
Directory 'Piplani' created successfully.
Enter subdirectory name: Bhumika
Directory 'Piplani/Bhumika' created successfully.

1. Create Directory
2. List Directories
3. Delete Directory
4. Exit
Enter your choice: 2
Enter directory path to list: Piplani
Directories in 'Piplani':
Bhumika

1. Create Directory
2. List Directories
3. Delete Directory
4. Exit
Enter your choice: 3
Enter directory name to delete: Piplani/Bhumika
Directory 'Piplani/Bhumika' deleted successfully.

1. Create Directory
2. List Directories
3. Delete Directory
4. Exit
Enter your choice: 4
```

- **Hierarchical Level Directory:**

```
#include <stdio.h>

#include <stdlib.h>

#include <sys/stat.h>

#include <unistd.h>

#include <dirent.h>

#include <string.h>


void createDirectory(const char *directoryName) {

    int status = mkdir(directoryName, 0777);


    if (status == 0) {

        printf("Directory '%s' created successfully.\n", directoryName);

    } else {

        printf("Failed to create directory '%s'.\n", directoryName);

    }

}


void listDirectories(const char *path) {

    DIR *dir;

    struct dirent *entry;


    if ((dir = opendir(path)) != NULL) {
        printf("Directories in '%s':\n", path);
        while ((entry = readdir(dir)) != NULL) {
            if (entry->d_type == DT_DIR && strcmp(entry->d_name, ".") != 0 && strcmp(entry->d_name, "..") != 0) {
                printf("%s\n", entry->d_name);
                char subpath[256];
                snprintf(subpath, sizeof(subpath), "%s/%s", path, entry->d_name);
                listDirectories(subpath);
            }
        }
        closedir(dir);
    } else {
        printf("Error opening directory '%s'.\n", path);
    }
}

void deleteDirectory(const char *directoryName) {
    if (rmdir(directoryName) == 0) {
        printf("Directory '%s' deleted successfully.\n", directoryName);
    } else {
        printf("Failed to delete directory '%s'.\n", directoryName);
    }
}
```

```
int main() {
    int choice;
    char dirName[100];
    char path[200];

    while (1) {
        printf("\n1. Create Directory\n");
        printf("2. List Directories\n");
        printf("3. Delete Directory\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter directory path to create: ");
                scanf("%s", path);
                createDirectory(path);
                break;
            case 2:
                printf("Enter directory path to list: ");
                scanf("%s", path);
                listDirectories(path);
                break;
            case 3:
                printf("Enter directory name to delete: ");
                scanf("%s", dirName);
                deleteDirectory(dirName);
                break;
            case 4:
                exit(0);
            default:
                printf("Invalid choice. Please enter a valid option.\n");
        }
    }

    return 0;
}
```

OUTPUT:



```
1. Create Directory
2. List Directories
3. Delete Directory
4. Exit
Enter your choice: 1
Enter directory path to create: Bhumika/Piplani
Directory 'Bhumika/Piplani' created successfully.

1. Create Directory
2. List Directories
3. Delete Directory
4. Exit
Enter your choice: 1
Enter directory path to create: Bhumika/Piplani/Btech
Directory 'Bhumika/Piplani/Btech' created successfully.

1. Create Directory
2. List Directories
3. Delete Directory
4. Exit
Enter your choice: 2
Enter directory path to list: Bhumika
Directories in 'Bhumika':
Piplani
Directories in 'Bhumika/Piplani':
Btech
Directories in 'Bhumika/Piplani/Btech':

1. Create Directory
2. List Directories
3. Delete Directory
4. Exit
Enter your choice: 3
Enter directory name to delete: Bhumika/Piplani/Btech
Directory 'Bhumika/Piplani/Btech' deleted successfully.

1. Create Directory
2. List Directories
3. Delete Directory
4. Exit
Enter your choice: 4
```