EXPERIMENT 1

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

THEORY -

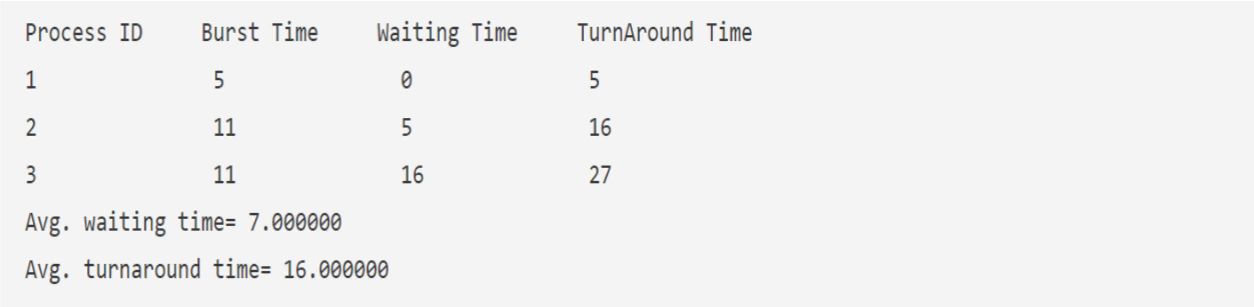
SOURCE CODE –

```
#include<stdio.h>
void findWaitingTime(int processes[], int n, int bt[], int wt[])
{
        wt [0] = 0;
        for (int i = 1; i < n; i++)
                wt[i] = bt[i-1] + wt[i-1];
}
void findTurnAroundTime(int processes[], int n, int bt[], int wt[], int tat[])
{
        for (int i = 0; i < n; i++)
                tat[i] = bt[i] + wt[i];
}
void findavgTime( int processes[], int n, int bt[])
{
        int wt[n], tat[n], total_wt = 0, total_tat = 0;
        findWaitingTime(processes, n, bt, wt);
        findTurnAroundTime(processes, n, bt, wt, tat);
        printf("Processes Burst time Waiting time Turn around time\n");
        for (int i=0; i<n; i++)
        {
                total_wt = total_wt + wt[i];
                total_tat = total_tat + tat[i];
                printf(" %d ",(i+1));
                printf(" %d ", bt[i] );
                printf(" %d",wt[i] );
                printf(" %d\n",tat[i]);
        }
        float s=(float)total_wt / (float)n;
        float t=(float)total_tat / (float)n;
        printf("Average waiting time = %f",s);
        printf("\n");
        printf("Average turn around time = %f ",t);
}
int main()
{
        int processes[] = {1, 2, 3};
        int n = sizeof processes / sizeof processes[0];

        //Burst time of all processes
        int burst_time[] = {10, 5, 8};

        findavgTime(processes, n, burst_time);
        return 0;
}
```

OUTPUT -



```
Process ID     Burst Time     Waiting Time     TurnAround Time
1               5               0               5
2               11              5               16
3               11              16              27
Avg. waiting time= 7.000000
Avg. turnaround time= 16.000000
```

EXPERIMENT 2

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

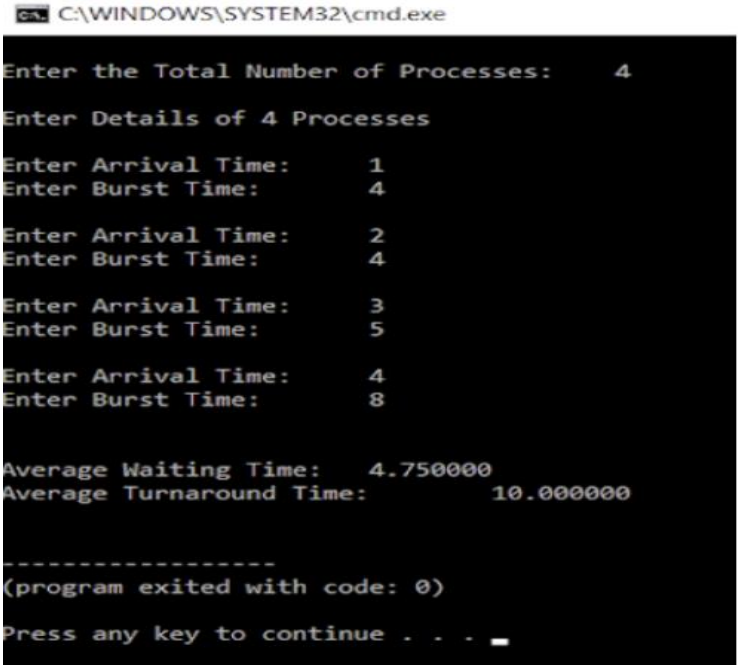
THEORY -

SOURCE CODE –

```
#include <stdio.h>


int main()
{
    int arrival_time[10], burst_time[10], temp[10];
    int i, smallest, count = 0, time, limit;
    double wait_time = 0, turnaround_time = 0, end;
    float average_waiting_time, average_turnaround_time;
    printf("nEnter the Total Number of Processes:t");
    scanf("%d", &limit);
    printf("nEnter Details of %d Processesn", limit);
    for(i = 0; i < limit; i++)
    {
        printf("nEnter Arrival Time:t");
        scanf("%d", &arrival_time[i]);
        printf("Enter Burst Time:t");
        scanf("%d", &burst_time[i]);
        temp[i] = burst_time[i];
    }
    burst_time[9] = 9999;
    for(time = 0; count != limit; time++)
    {
        smallest = 9;
        for(i = 0; i < limit; i++)
        {
            if(arrival_time[i] <= time && burst_time[i] < burst_time[smallest] &&
burst_time[i] > 0)
            {
                smallest = i;
            }
        }
        burst_time[smallest]--;
        if(burst_time[smallest] == 0)
        {
            count++;
            end = time + 1;
            wait_time = wait_time + end - arrival_time[smallest] - temp[smallest];
            turnaround_time = turnaround_time + end - arrival_time[smallest];
        }
    }
    average_waiting_time = wait_time / limit;
    average_turnaround_time = turnaround_time / limit;
    printf("nnAverage Waiting Time:t%lfn", average_waiting_time);
    printf("Average Turnaround Time:t%lfn", average_turnaround_time);
    return 0;
}
```

OUTPUT-



```
Enter the Total Number of Processes:    4

Enter Details of 4 Processes

Enter Arrival Time:     1
Enter Burst Time:       4

Enter Arrival Time:     2
Enter Burst Time:       4

Enter Arrival Time:     3
Enter Burst Time:       5

Enter Arrival Time:     4
Enter Burst Time:       8


Average Waiting Time:   4.750000
Average Turnaround Time:        10.000000


------------------
(program exited with code: 0)

Press any key to continue . . .
```

EXPERIMENT 3

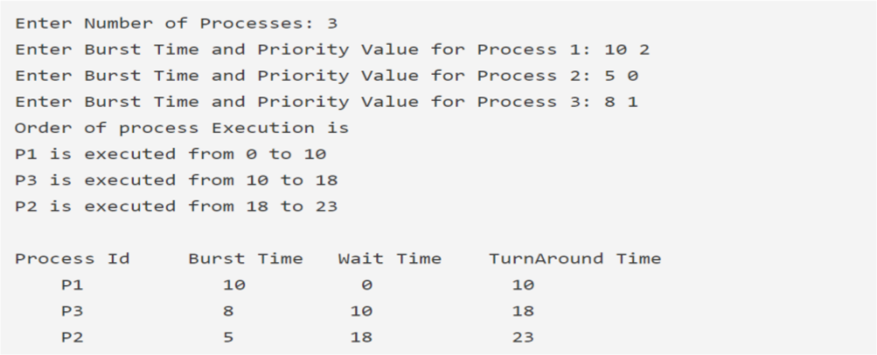
AIM – Write a program to perform priority scheduling.

THOERY –

SOURCE CODE-

```
#include <stdio.h>
void swap(int *a,int *b)
{
    int temp=*a;
    *a=*b;
    *b=temp;
}
int main()
{
    int n;
    printf("Enter Number of Processes: ");
    scanf("%d",&n);
    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;
        for(int j=i;j<n;j++)
        {
            if(p[j] > a)
            {
                a=p[j];
                m=j;
            }
        }
        swap(&p[i], &p[m]);
        swap(&b[i], &b[m]);
        swap(&index[i],&index[m]);
    }
    int t=0;
    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 –



```
Enter Number of Processes: 3
Enter Burst Time and Priority Value for Process 1: 10 2
Enter Burst Time and Priority Value for Process 2: 5 0
Enter Burst Time and Priority Value for Process 3: 8 1
Order of process Execution is
P1 is executed from 0 to 10
P3 is executed from 10 to 18
P2 is executed from 18 to 23

Process Id     Burst Time   Wait Time    TurnAround Time
    P1            10           0            10
    P3            8            10           18
    P2            5            18           23
```

EXPERIMENT 4

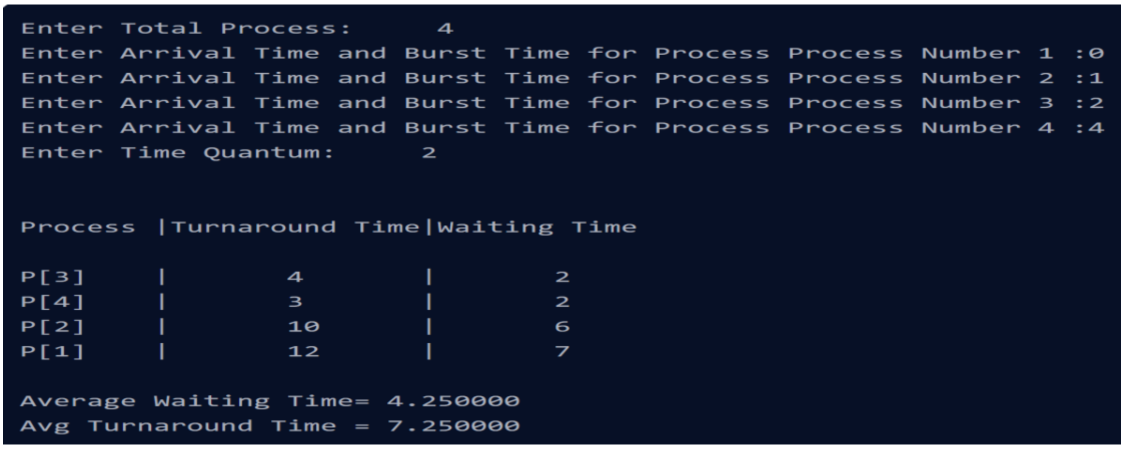
AIM – Write a program to implement CPU scheduling for Round Robin.

THEORY –

SOURCE CODE –

```
#include<stdio.h>
 int main()
{
  int cnt,j,n,t,remain,flag=0,tq;
  int wt=0,tat=0,at[10],bt[10],rt[10];
  printf("Enter Total Process:\t ");
  scanf("%d",&n);
  remain=n;
  for(cnt=0;cnt<n;cnt++)
  {
    printf("Enter Arrival Time and Burst Time for Process Process Number %d :",cnt+1);
    scanf("%d",&at[cnt]);
    scanf("%d",&bt[cnt]);
    rt[cnt]=bt[cnt];
  }
  printf("Enter Time Quantum:\t");
  scanf("%d",&tq);
  printf("\n\nProcess\t|Turnaround Time|Waiting Time\n\n");
  for(t=0,cnt=0;remain!=0;)
  {
    if(rt[cnt]<=tq && rt[cnt]>0)
    {
      t+=rt[cnt];
      rt[cnt]=0;
      flag=1;
    }
    else if(rt[cnt]>0)
    {
      rt[cnt]-=tq;
      t+=tq;
     }
     if(rt[cnt]==0 && flag==1)
     {
      remain--;
      printf("P[%d]\t|\t%d\t|\t%d\n",cnt+1,t-at[cnt],t-at[cnt]-bt[cnt]);
      wt+=t-at[cnt]-bt[cnt];
      tat+=t-at[cnt];
      flag=0;
     }
     if(cnt==n-1)
      cnt=0;
     else if(at[cnt+1]<=t)
      cnt++;
     else
      cnt=0;
    }
    printf("\nAverage Waiting Time= %f\n",wt*1.0/n);
    printf("Avg Turnaround Time = %f",tat*1.0/n);
    return 0;
  }
```

OUTPUT-



```
Enter Total Process:      4
Enter Arrival Time and Burst Time for Process Process Number 1 :0
Enter Arrival Time and Burst Time for Process Process Number 2 :1
Enter Arrival Time and Burst Time for Process Process Number 3 :2
Enter Arrival Time and Burst Time for Process Process Number 4 :4
Enter Time Quantum:       2


Process |Turnaround Time|Waiting Time

P[3]    |       4       |       2
P[4]    |       3       |       2
P[2]    |       10      |       6
P[1]    |       12      |       7

Average Waiting Time= 4.250000
Avg Turnaround Time = 7.250000
```

EXPERIMENT 5

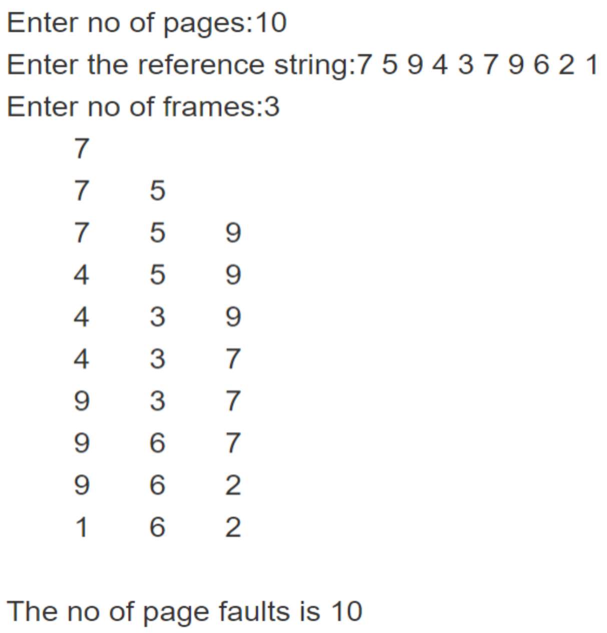
AIM – a) Write a program to illustrate Least Recent Used Page replacement algorithm.

THEORY –

SOURCE CODE –

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

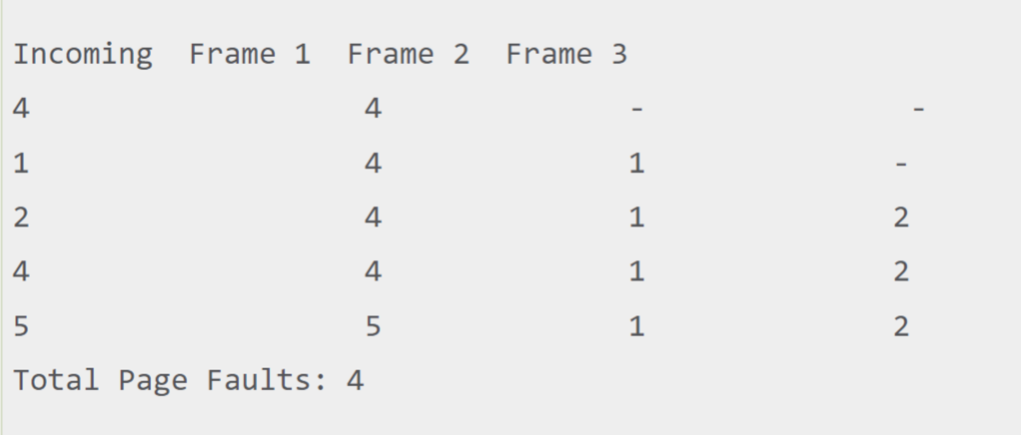
AIM – b)Write a program for page replacement policy using first in first out algorithm.

THEORY –

SOURCE 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 –



```
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
```

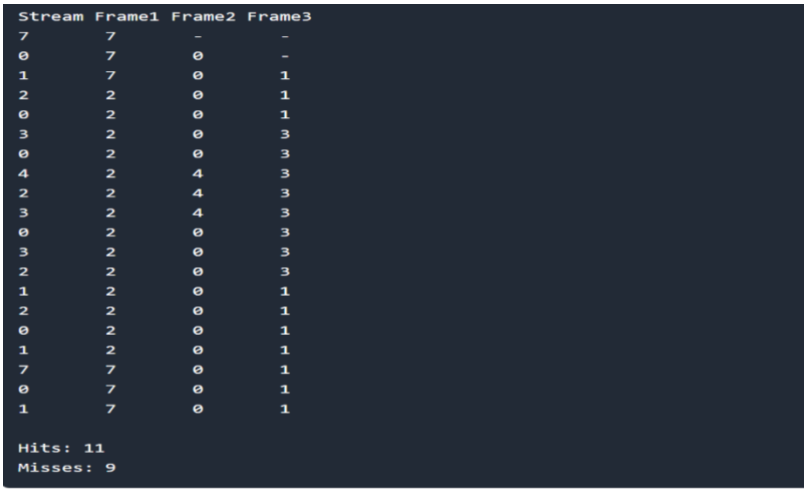
AIM- c) Write a program to illustrate Optimal Page Replacement algorithm.

THEORY –

SOURCE 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)
{
    // initially none of the frames are occupied
    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-



```
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.

THEORY –

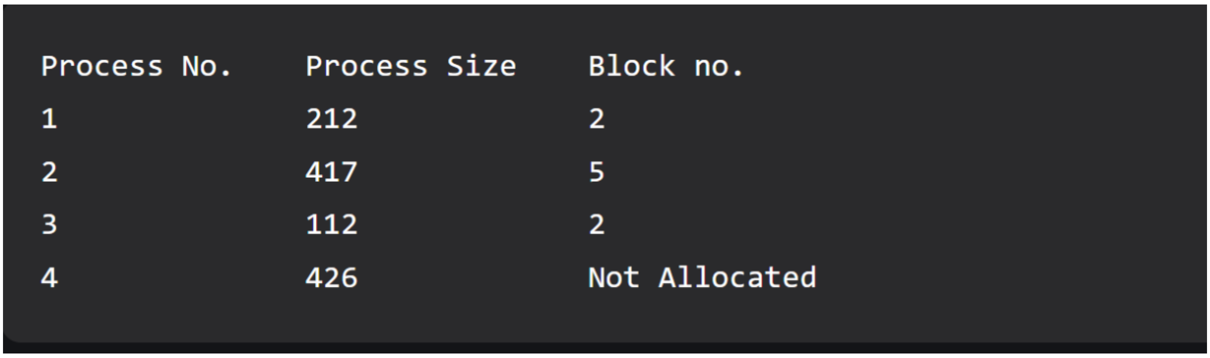
SOURCE CODE-

FIRST FIT

```
// C implementation of First - Fit algorithm
#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++)    //here, n -> number of processes
        {
                for (j = 0; j < m; j++)   //here, m -> number of blocks
                {
                        if (blockSize[j] >= processSize[i])
                        {
                                // allocating block j to the ith process
                                allocation[i] = j;

                                blockSize[j] -= processSize[i];
                                break; //go to the next process in the queue
                        }
                }
        }
        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; //number of blocks in the memory
        int n; //number of processes in the input queue
        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-

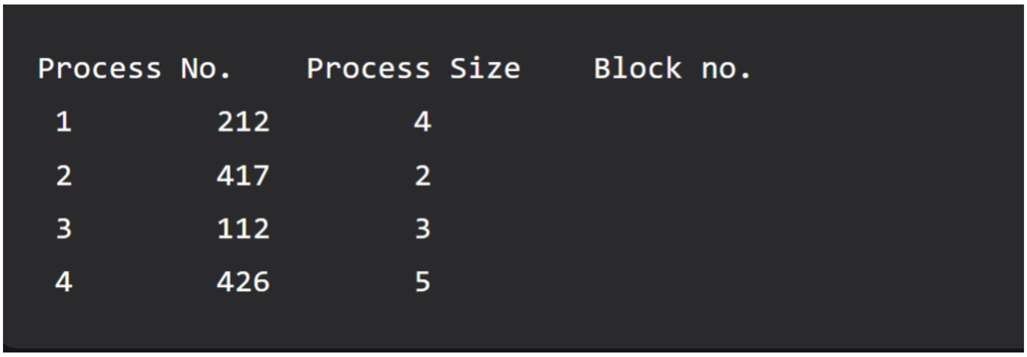


```
Process No.    Process Size    Block no.
1              212             2
2              417             5
3              112             2
4              426             Not Allocated
```

BEST FIT

```
#include<iostream>
using namespace std;
void bestFit(int blockSize[], int m, int processSize[], int n)
{
        int allocation[n];
        for (int i = 0; i < n; i++)
                allocation[i] = -1;
        for (int i = 0; i < n; i++)
        {
                // Find the best fit block for current process
                int bestIdx = -1;
                for (int j = 0; j < m; j++)
                {
                        if (blockSize[j] >= processSize[i])
                        {
                                if (bestIdx == -1)
                                        bestIdx = j;
                                else if (blockSize[bestIdx] > blockSize[j])
                                        bestIdx = j;
                        }
                }
                if (bestIdx != -1)
                {
                        allocation[i] = bestIdx;
                        blockSize[bestIdx] -= processSize[i];
                }
        }
        cout << "\nProcess No.\tProcess Size\tBlock no.\n";
        for (int i = 0; i < n; i++)
        {
                cout << " " << i+1 << "\t\t" << processSize[i] << "\t\t";
                if (allocation[i] != -1)
                        cout << allocation[i] + 1;
                else
                        cout << "Not Allocated";
                cout << endl;
        }
}
int main()
{
        int blockSize[] = {100, 500, 200, 300, 600};
        int processSize[] = {212, 417, 112, 426};
        int m = sizeof(blockSize) / sizeof(blockSize[0]);
        int n = sizeof(processSize) / sizeof(processSize[0]);
        bestFit(blockSize, m, processSize, n);
        return 0 ;
}
```

OUTPUT –

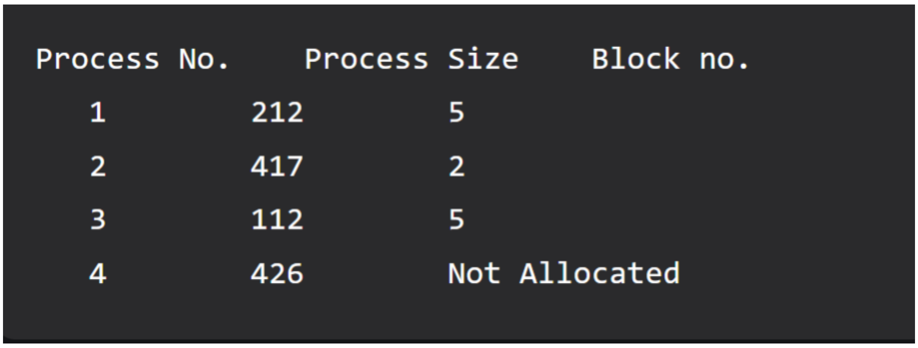


```
Process No.    Process Size    Block no.
 1        212        4
 2        417        2
 3        112        3
 4        426        5
```

WORST FIT

```
#include<bits/stdc++.h>
using namespace std;
void worstFit(int blockSize[], int m, int processSize[], int n)
{
        int allocation[n];
        memset(allocation, -1, sizeof(allocation));
        for (int i=0; i<n; i++)
        {
                // Find the best fit block for current process
                int wstIdx = -1;
                for (int j=0; j<m; j++)
                {
                        if (blockSize[j] >= processSize[i])
                        {
                                if (wstIdx == -1)
                                        wstIdx = j;
                                else if (blockSize[wstIdx] < blockSize[j])
                                        wstIdx = j;
                        }
                }
                if (wstIdx != -1)
                {
                        allocation[i] = wstIdx;
                        blockSize[wstIdx] -= processSize[i];
                }
        }
        cout << "\nProcess No.\tProcess Size\tBlock no.\n";
        for (int i = 0; i < n; i++)
        {
                cout << " " << i+1 << "\t\t" << processSize[i] << "\t\t";
                if (allocation[i] != -1)
                        cout << allocation[i] + 1;
                else
                        cout << "Not Allocated";
                cout << endl;
        }
}
int main()
{
        int blockSize[] = {100, 500, 200, 300, 600};
        int processSize[] = {212, 417, 112, 426};
        int m = sizeof(blockSize)/sizeof(blockSize[0]);
        int n = sizeof(processSize)/sizeof(processSize[0]);
        worstFit(blockSize, m, processSize, n);
        return 0 ;
}
```

OUTPUT-



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

EXPERIMENT 7

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

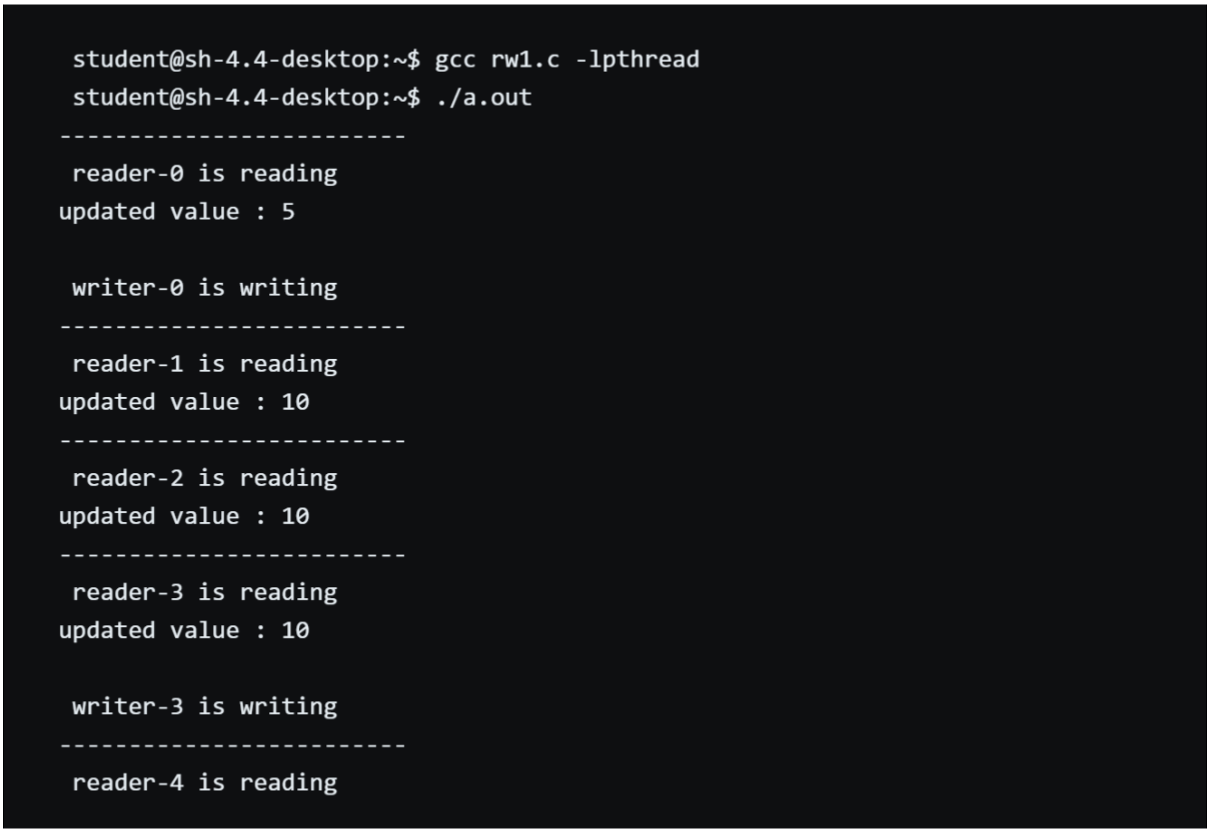
THEORY-

SOURCE CODE-

```
#include<semaphore.h>
#include<stdio.h>
#include<pthread.h>
# include<bits/stdc++.h>
using namespace std;
void *reader(void *);
void *writer(void *);
int readcount=0,writecount=0,sh_var=5,bsize[5];
sem_t x,y,z,rsem,wsem;
pthread_t r[3],w[2];
void *reader(void *i)
{
    cout << "\n-------------------------";
    cout << "\n\n reader-" << i << " is reading";
    sem_wait(&z);
    sem_wait(&rsem);
    sem_wait(&x);
    readcount++;
    if(readcount==1)
      sem_wait(&wsem);
    sem_post(&x);
    sem_post(&rsem);
    sem_post(&z);
    cout << "\nupdated value :" << sh_var;
    sem_wait(&x);
    readcount--;
    if(readcount==0)
      sem_post(&wsem);
    sem_post(&x);
```

```
}
void *writer(void *i)
{
    cout << "\n\n writer-" << i << "is writing";
    sem_wait(&y);
    writecount++;
    if(writecount==1)
    sem_wait(&rsem);
    sem_post(&y);
    sem_wait(&wsem);
    sh_var=sh_var+5;
    sem_post(&wsem);
    sem_wait(&y);
    writecount--;
    if(writecount==0)
    sem_post(&rsem);
    sem_post(&y);
}
int main()
{
    sem_init(&x,0,1);
    sem_init(&wsem,0,1);
    sem_init(&y,0,1);
    sem_init(&z,0,1);
    sem_init(&rsem,0,1);
    pthread_create(&r[0],NULL,(void *)reader,(void *)0);
    pthread_create(&w[0],NULL,(void *)writer,(void *)0);
    pthread_create(&r[1],NULL,(void *)reader,(void *)1);
    pthread_create(&r[2],NULL,(void *)reader,(void *)2);
    pthread_create(&r[3],NULL,(void *)reader,(void *)3);
    pthread_create(&w[1],NULL,(void *)writer,(void *)3);
    pthread_create(&r[4],NULL,(void *)reader,(void *)4);
    pthread_join(r[0],NULL);
    pthread_join(w[0],NULL);
    pthread_join(r[1],NULL);
    pthread_join(r[2],NULL);
    pthread_join(r[3],NULL);
    pthread_join(w[1],NULL);
    pthread_join(r[4],NULL);
    return(0);
}
```

OUTPUT –



```
 student@sh-4.4-desktop:~$ gcc rw1.c -lpthread
 student@sh-4.4-desktop:~$ ./a.out
-------------------------
 reader-0 is reading
updated value : 5

 writer-0 is writing
-------------------------
 reader-1 is reading
updated value : 10
-------------------------
 reader-2 is reading
updated value : 10
-------------------------
 reader-3 is reading
updated value : 10

 writer-3 is writing
-------------------------
 reader-4 is reading
```

EXPERIMENT 8

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

THEORY-

SOURCE CODE-

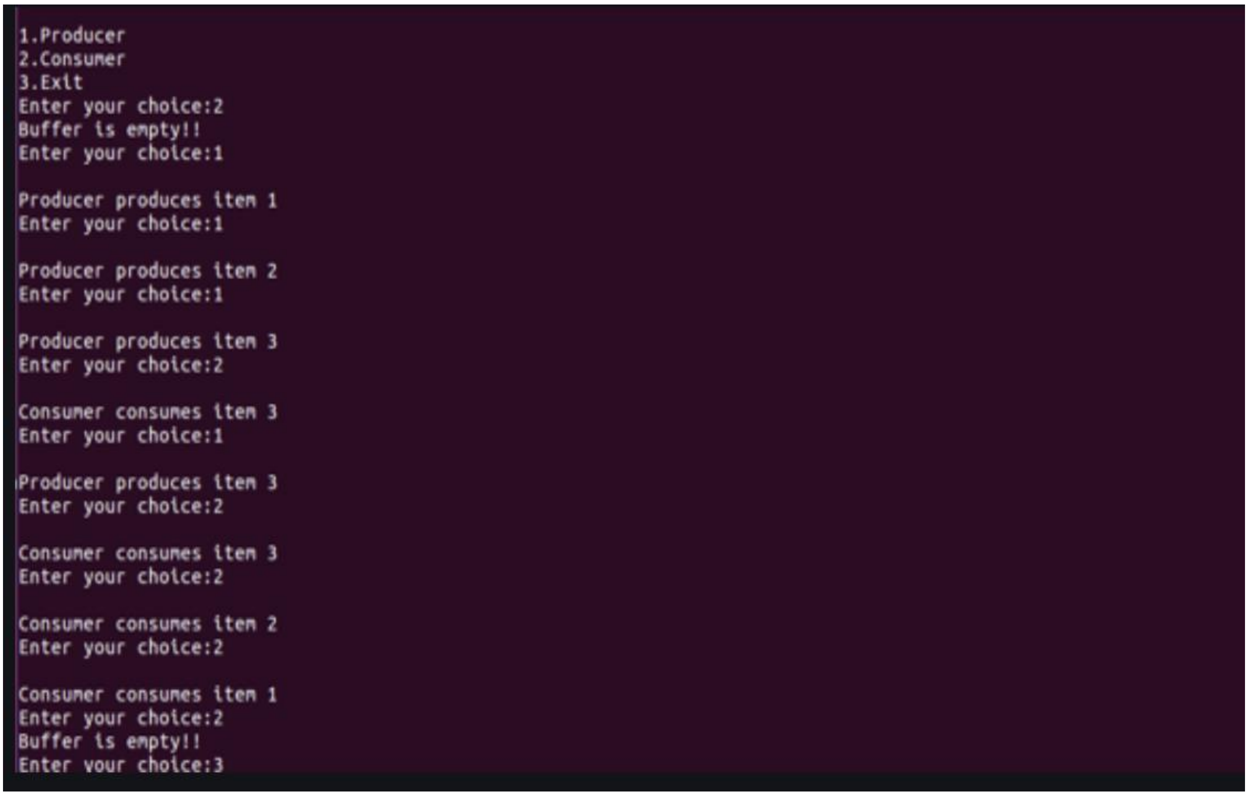
```
#include <stdio.h>
#include <stdlib.h>
int full = 0;
int empty = 10, x = 0;
void producer()
{
        --mutex;
        ++full;
        --empty;
        x++;
        printf("\nProducer produces"
                "item %d",
                x);
        ++mutex;
}
void consumer()
{
        --mutex;
        --full;
        ++empty;
        printf("\nConsumer consumes "
                "item %d",
                x);
        x--;

        // Increase mutex value by 1
        ++mutex;
}
int main()
```

```
{
        int n, i;
        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 (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!");
                        }
                case 3:
                        exit(0);
                        break;
                }
        }
}
```

OUTPUT-



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

Producer produces item 1
Enter your choice:1

Producer produces item 2
Enter your choice:1

Producer produces item 3
Enter your choice:2

Consumer consumes item 3
Enter your choice:1

Producer produces item 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
```

EXPERIMENT 8

AIM – To implement Bankers algorithm for deadlock avoidance.

THEORY-

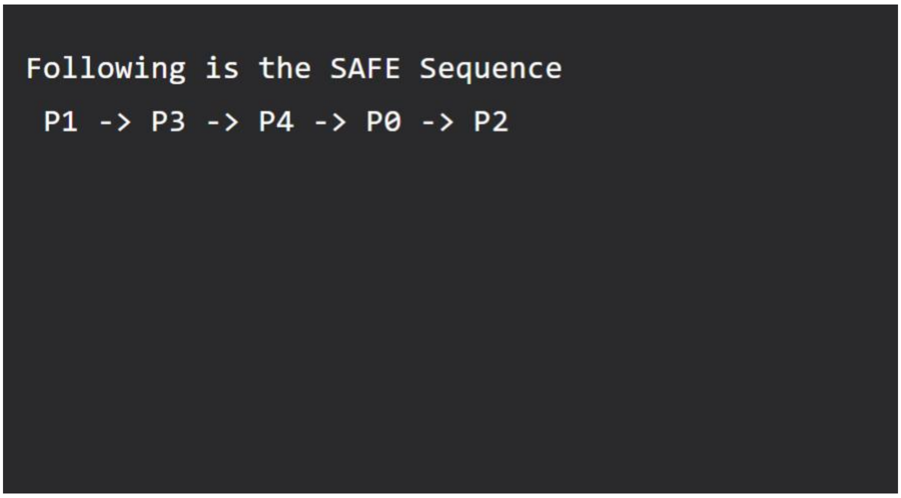
SOURCE CODE-

```
#include <stdio.h>
int main()
{
        int n, m, i, j, k;
        n = 5; // Number of processes
        m = 3; // Number of resources
        int alloc[5][3] = { { 0, 1, 0 }, // P0
                                      { 2, 0, 0 }, // P1
                                      { 3, 0, 2 }, // P2
                                      { 2, 1, 1 }, // P3
                                      { 0, 0, 2 } }; // P4
        int max[5][3] = { { 7, 5, 3 },
                                { 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;
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-



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