LABORATORY REPORT

OPERATING SYSTEM

Submitted by

Name: Sidharth Mukherjee

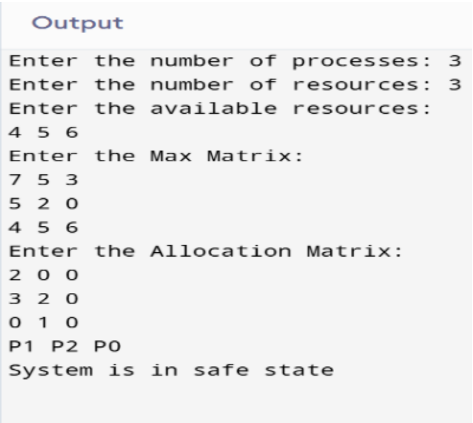
Reg. No.: 23BET10003

SCHOOL OF COMPUTING SCIENCE AND ENGINEERING

VIT BHOPAL UNIVERSITY

Q1 Bankers Algorithm

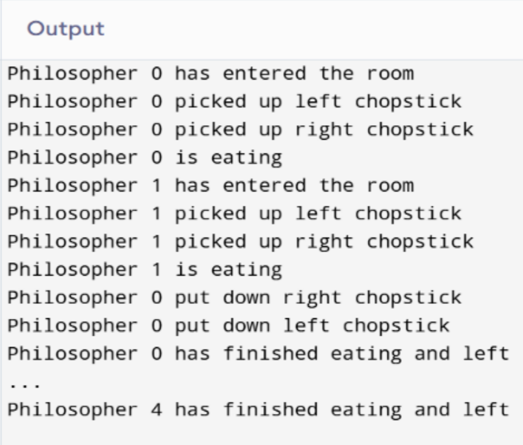
```
#include <stdio.h>
#include <conio.h>
void main() {
        int
k=0,a=0,b=0,instance[5],availability[5],allocated[10][5],need[10][5],MAX[10][5],process,P[10],no
_of_resources, cnt=0,i, j;
        printf("\n Enter the number of resources : ");
        scanf("%d", &no_of_resources);
        printf("\n enter the max instances of each resources\n");
        for (i=0;i<no_of_resources;i++) {
                availability[i]=0;
                printf("%c= ",(i+97));
                scanf("%d",&instance[i]);
        }
        printf("\n Enter the number of processes : ");
        scanf("%d", &process);
        printf("\n Enter the allocation matrix \n     ");
        for (i=0;i<no_of_resources;i++)
        printf(" %c",(i+97));
        printf("\n");
        for (i=0;i <process;i++) {
                P[i]=i;
                printf("P[%d]  ",P[i]);
                for (j=0;j<no_of_resources;j++) {
                        scanf("%d",&allocated[i][j]);
                        availability[j]+=allocated[i][j];
                }
        }
        printf("\nEnter the MAX matrix \n     ");
        for (i=0;i<no_of_resources;i++) {
                printf(" %c",(i+97));
                availability[i]=instance[i]-availability[i];
        }
        printf("\n");
        for (i=0;i <process;i++) {
                printf("P[%d]  ",i);
                for (j=0;j<no_of_resources;j++)
                 scanf("%d", &MAX[i][j]);
        }
        printf("\n");
        A: a=-1;
        for (i=0;i <process;i++) {
                cnt=0;
                b=P[i];
                for (j=0;j<no_of_resources;j++) {
                        need[b][j] = MAX[b][j]-allocated[b][j];
                        if(need[b][j]<=availability[j])
                         cnt++;
                }
                if(cnt==no_of_resources) {
                        op[k++]=P[i];
                        for (j=0;j<no_of_resources;j++)
                        availability[j]+=allocated[b][j];
                } else
                 P[++a]=P[i];
        }
        if(a!=-1) {
                process=a+1;
                goto A;
        }
        printf("\t <");
        for (i=0;i<k;i++)
        printf(" P[%d] ",op[i]);
        printf(">");
        getch();
}
```



```
Enter the number of processes: 3
Enter the number of resources: 3
Enter the available resources:
4 5 6
Enter the Max Matrix:
7 5 3
5 2 0
4 5 6
Enter the Allocation Matrix:
2 0 0
3 2 0
0 1 0
P1 P2 P0
System is in safe state
```

Q2 Dining Philosopher

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

sem_t room;           // counting semaphore
sem_t chopstick[5];   // binary semaphore

void * philosopher(void *);
void eat(int);

void eat(int phil)
{
       printf("\nPhilosopher %d is eating",phil);
}

int main()
{
       int i,a[5];
       pthread_t tid[5];       // creation of threads refering to 5 philosophers

       sem_init(&room,0,4);   // initializations of semaphore  varring from 0 to 4.

       for(i=0;i<5;i++)
              sem_init(&chopstick[i],0,1);  //initializations of binary semaphore .

       for(i=0;i<5;i++){
              a[i]=i;
              pthread_create(&tid[i],NULL,philosopher,(void *)&a[i]);  // creation of
philosopher and assigning it a number.
       }
       for(i=0;i<5;i++)
              pthread_join(tid[i],NULL);  // waits until a thread gets terminated
}

void * philosopher(void * num)
{
       int phil=*(int *)num;
       sem_wait(&room);                          // semaphore function to checks if
resources are available.
       printf("\nPhilosopher %d has entered room",phil);
       sem_wait(&chopstick[phil]);               // semaphore function to checks if
chopstick is available.
       sem_wait(&chopstick[(phil+1)%5]);
       eat(phil);
       sleep(2);
       printf("\nPhilosopher %d has finished eating",phil);
       sem_post(&chopstick[(phil+1)%5]);   // gives confirmation if semophore is
released successfully
       sem_post(&chopstick[phil]);
       sem_post(&room);
}
```

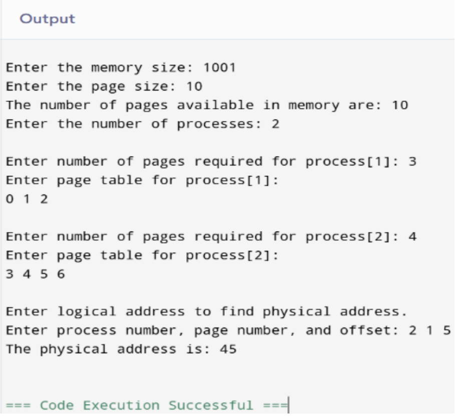


Output

```
Philosopher 0 has entered the room
Philosopher 0 picked up left chopstick
Philosopher 0 picked up right chopstick
Philosopher 0 is eating
Philosopher 1 has entered the room
Philosopher 1 picked up left chopstick
Philosopher 1 picked up right chopstick
Philosopher 1 is eating
Philosopher 0 put down right chopstick
Philosopher 0 put down left chopstick
Philosopher 0 has finished eating and left
...
Philosopher 4 has finished eating and left
```

Q3 Paging

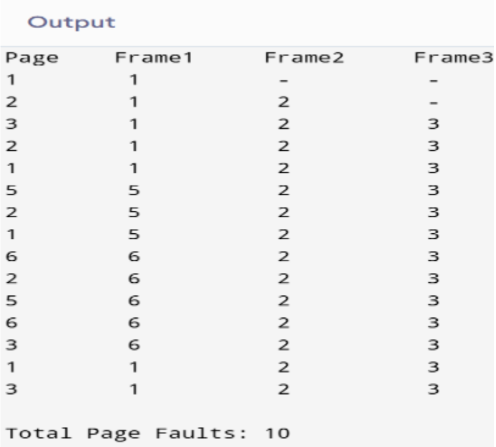
```
#include<stdio.h>
#include<conio.h>
main()
{
 int ms, ps, nop, np, rempages, i, j, x, y, pa, offset;
 int s[10], fno[10][20];

clrscr();

printf("\nEnter the memory size -- ");
scanf("%d",&ms);

printf("\nEnter the page size -- ");
scanf("%d",&ps);

nop = ms/ps;
printf("\nThe no. of pages available in memory are -- %d ",nop);

printf("\nEnter number of processes -- ");
 scanf("%d",&np);
rempages = nop;
for(i=1;i<=np;i++)

{

printf("\nEnter no. of pages required for p[%d]-- ",i);
 scanf("%d",&s[i]);

if(s[i] >rempages)
{

printf("\nMemory is Full");
break;
}
rempages = rempages - s[i];

printf("\nEnter pagetable for p[%d] --- ",i);
 for(j=0;j<s[i];j++)
scanf("%d",&fno[i][j]);
}

printf("\nEnter Logical Address to find Physical Address ");
printf("\nEnter process no. and pagenumber and offset -- ");

scanf("%d %d %d",&x,&y, &offset);



if(x>np || y>=s[i] || offset>=ps)
printf("\nInvalid Process or Page Number or offset");

else
{ pa=fno[x][y]*ps+offset;
printf("\nThe Physical Address is -- %d",pa);

}
getch();
}
```



```
Enter the memory size: 1001
Enter the page size: 10
The number of pages available in memory are: 10
Enter the number of processes: 2

Enter number of pages required for process[1]: 3
Enter page table for process[1]:
0 1 2

Enter number of pages required for process[2]: 4
Enter page table for process[2]:
3 4 5 6

Enter logical address to find physical address.
Enter process number, page number, and offset: 2 1 5
The physical address is: 45


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

Q4 Page Replacement

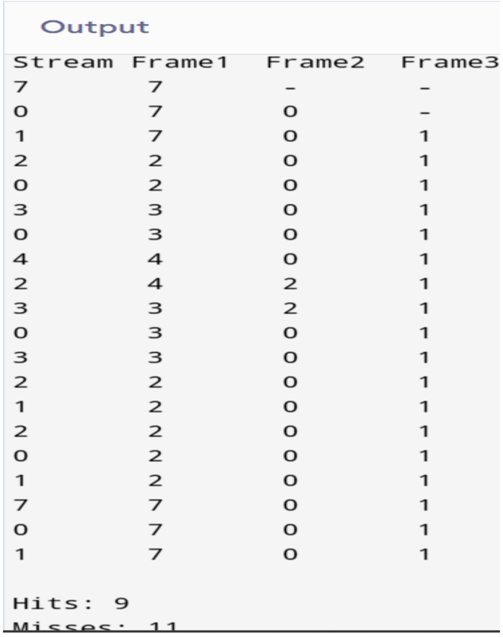
```c
#include<stdio.h>
int main()
{
    int incomingStream[] = {4, 1, 2, 4, 5};
    int pageFaults = 0;
    int frames = 3;
    int m, n, s, pages;

    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

Page      Frame1      Frame2      Frame3
1          1           -           -
2          1           2           -
3          1           2           3
2          1           2           3
1          1           2           3
5          5           2           3
2          5           2           3
1          5           2           3
6          6           2           3
2          6           2           3
5          6           2           3
6          6           2           3
3          6           2           3
1          1           2           3
3          1           2           3

Total Page Faults: 10
```

Q4 Optimal Page Replacement

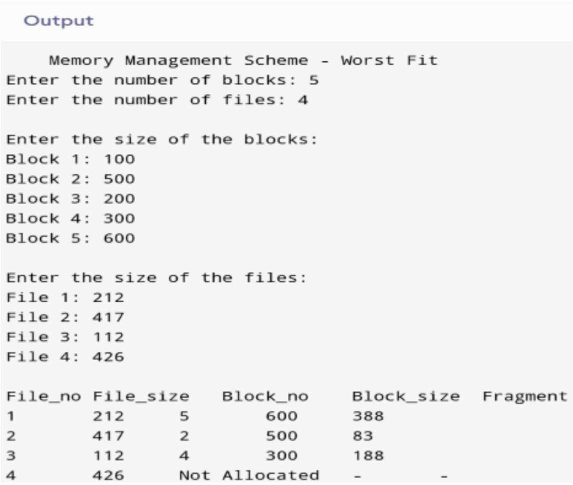
```c
#include <stdio.h>

// This function checks if current strea item(key) exists in any of the frames or not
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){

    // print current reference stream item
    printf("\n%d \t\t", item);

    // print frame occupants one by one
    for(int i = 0; i < max_frames; i++){
        if(i < frame_occupied)
            printf("%d \t\t", frame_items[i]);
        else
            printf("- \t\t");
    }
}
// This Function helps in finding frame that will not be used
// for the longest period of time in future in ref_str[0 ... refStrLen - 1]
int predict(int ref_str[], int frame_items[], int refStrLen, int index, int
frame_occupied)
{
    // For each current occupant in frame item
    // we try to find the frame item that will not be referenced in
    // for the longest in future in the upcoming reference string
    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 we find a page that is never referenced in future,
        // return it immediately as its the best
        if (j == refStrLen)
            return i;
    }

    // If none of the frame items appear in reference string
    // in the future then we return 0th index. Otherwise we return result
    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);

    // Here we traverse through reference string
    // and check for miss and hit.
    int hits = 0;
    for (int i = 0; i < refStrLen; i++) {

        // If found already in the frame items : HIT
        if (search(ref_str[i], frame_items, frame_occupied)) {
            hits++;
            printCurrFrames(ref_str[i], frame_items, frame_occupied, max_frames);
            continue;
        }

        // If not found in frame items : MISS

        // If frames are empty then current reference string item in frame
        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 we need to use optmial algorithm to find
        // frame index where we need to do replacement for this
        // incoming reference string item
        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);
}

// Driver Function
int main()
{
    // int ref_str[] = {9, 0, 5, 1, 0, 3, 0, 4, 1, 3, 0, 3, 1, 3};
    int ref_str[] = {7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1};
    int refStrLen = sizeof(ref_str) / sizeof(ref_str[0]);
    int max_frames = 3;
    int frame_items[max_frames];

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



```
Stream Frame1  Frame2  Frame3
7       7       -       -
0       7       0       -
1       7       0       1
2       2       0       1
0       2       0       1
3       3       0       1
0       3       0       1
4       4       0       1
2       4       2       1
3       3       2       1
0       3       0       1
3       3       0       1
2       2       0       1
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: 9
Misses: 11
```

Q5 Dynamic Allocation

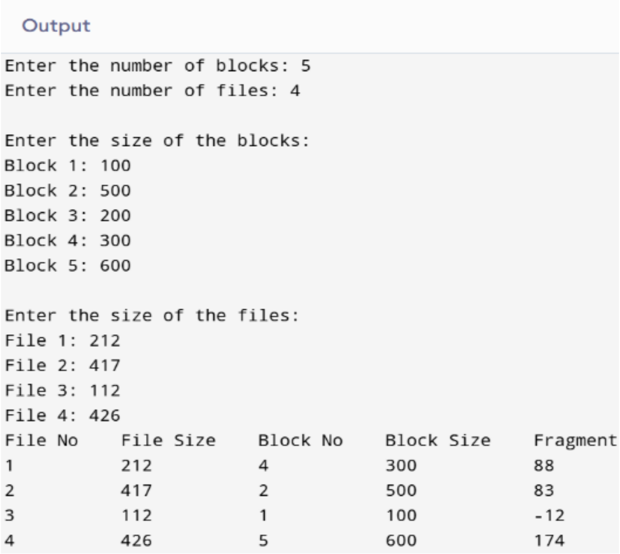
```
#include<stdio.h>
#include<conio.h>
#define max 25
void main()
{
int frag[max],b[max],f[max],i,j,nb,nf,temp;
static int bf[max],ff[max];
clrscr();
printf("\n\tMemory Management Scheme - First Fit");
printf("\nEnter the number of blocks:");
scanf("%d",&nb);
printf("Enter the number of files:");
scanf("%d",&nf);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



```
    Memory Management Scheme - Worst Fit
Enter the number of blocks: 5
Enter the number of files: 4

Enter the size of the blocks:
Block 1: 100
Block 2: 500
Block 3: 200
Block 4: 300
Block 5: 600

Enter the size of the files:
File 1: 212
File 2: 417
File 3: 112
File 4: 426

File_no File_size   Block_no     Block_size  Fragment
1        212       5        600       388
2        417       2        500       83
3        112       4        300       188
4        426      Not Allocated    -        -
```

```
{
temp=b[j]-f[i];
if(temp>=0)
{
ff[i]=j;
break;
}
}
}
frag[i]=temp;
bf[ff[i]]=1;
}
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]);
getch();
}
```

```
#include<stdio.h>
#include<conio.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];
clrscr();
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");
```

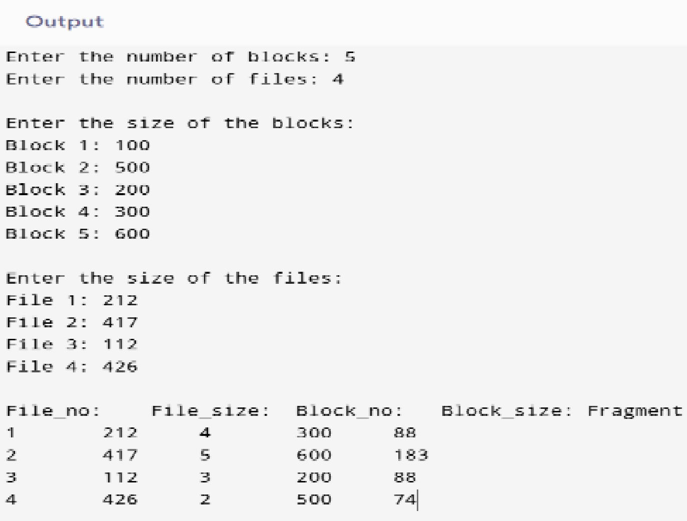


Output

```
Enter the number of blocks: 5
Enter the number of files: 4

Enter the size of the blocks:
Block 1: 100
Block 2: 500
Block 3: 200
Block 4: 300
Block 5: 600

Enter the size of the files:
File 1: 212
File 2: 417
File 3: 112
File 4: 426
File No     File Size     Block No     Block Size     Fragment
1           212           4            300            88
2           417           2            500            83
3           112           1            100            -12
4           426           5            600            174
```

```c
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]);
getch();
}
```

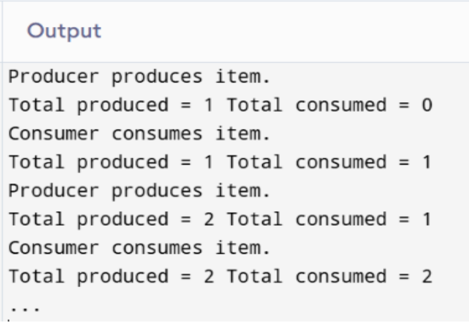
```c
#include<stdio.h>
#include<conio.h>
#define max 25
void main()
{
int frag[max],b[max],f[max],i,j,nb,nf,temp,highest=0;
static int bf[max],ff[max];
clrscr();
printf("\n\tMemory Management Scheme - Worst Fit");
printf("\nEnter the number of blocks:");
scanf("%d",&nb);
printf("Enter the number of files:");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



```
Enter the number of blocks: 5
Enter the number of files: 4

Enter the size of the blocks:
Block 1: 100
Block 2: 500
Block 3: 200
Block 4: 300
Block 5: 600

Enter the size of the files:
File 1: 212
File 2: 417
File 3: 112
File 4: 426

File_no:    File_size:  Block_no:   Block_size: Fragment
1        212      4        300      88
2        417      5        600      183
3        112      3        200      88
4        426      2        500      74
```

Q7 Producer Consumer Problem

```
#include&lt;bits/stdc++.h&gt;
#include&lt;pthread.h&gt;
#include&lt;semaphore.h&gt;
#include &lt;unistd.h&gt;
using namespace std;

// Declaration
int r1,total_produced=0,total_consume=0;

// Semaphore declaration
sem_t notEmpty;

// Producer Section
void* produce(void *arg){
    while(1){
      cout&lt;&lt;&quot;Producer produces item.&quot;&lt;&lt;endl;
      cout&lt;&lt;&quot;Total produced = &quot;&lt;&lt;++total_produced&lt;&lt;
        &quot; Total consume = &quot;&lt;&lt;total_consume*-1&lt;&lt;endl;
      sem_post(¬Empty);
      sleep(rand()%100*0.01);
    }
}

// Consumer Section
void* consume(void *arg){
    while(1){
      sem_wait(¬Empty);
      cout&lt;&lt;&quot;Consumer consumes item.&quot;&lt;&lt;endl;
      cout&lt;&lt;&quot;Total produced = &quot;&lt;&lt;total_produced&lt;&lt;
        &quot; Total consume = &quot;&lt;&lt;(--total_consume)*-1&lt;&lt;endl;
      sleep(rand()%100*0.01);
    }
}

int main(int argv,char *argc[]){

    // thread declaration
    pthread_t producer,consumer;

    // Declaration of attribute......

    pthread_attr_t attr;

    // semaphore initialization
    sem_init(¬Empty,0,0);

    // pthread_attr_t initialization
    pthread_attr_init(&amp;attr);
    pthread_attr_setdetachstate(&amp;attr,PTHREAD_CREATE_JOINABLE);

    // Creation of process
    r1=pthread_create(&amp;producer,&amp;attr,produce,NULL);
    if(r1){
      cout&lt;&lt;&quot;Error in creating thread&quot;&lt;&lt;endl;
      exit(-1);
    }

    r1=pthread_create(&amp;consumer,&amp;attr,consume,NULL);
    if(r1){
      cout&lt;&lt;&quot;Error in creating thread&quot;&lt;&lt;endl;
      exit(-1);
    }

    // destroying the pthread_attr
    pthread_attr_destroy(&amp;attr);

    // Joining the thread
    r1=pthread_join(producer,NULL);
    if(r1){
      cout&lt;&lt;&quot;Error in joining thread&quot;&lt;&lt;endl;
      exit(-1);
```



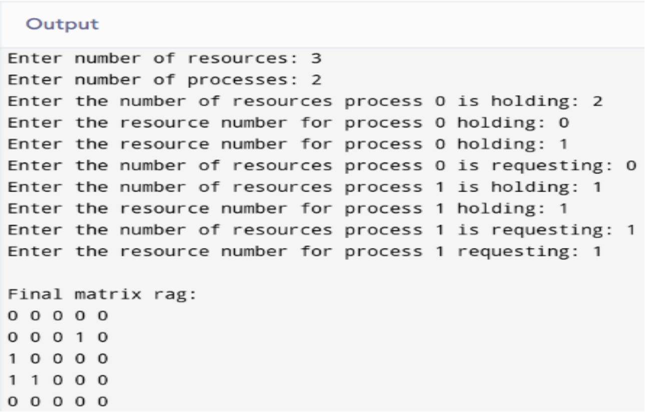
Output

```
Producer produces item.
Total produced = 1 Total consumed = 0
Consumer consumes item.
Total produced = 1 Total consumed = 1
Producer produces item.
Total produced = 2 Total consumed = 1
Consumer consumes item.
Total produced = 2 Total consumed = 2
...
```

```
    }

    r1=pthread_join(consumer,NULL);
    if(r1){
      cout&lt;&lt;&quot;Error in joining thread&quot;&lt;&lt;endl;
      exit(-1);
    }

    // Exiting thread
    pthread_exit(NULL);

    return 0;
}
```

Q8 Resource Allocation Graph

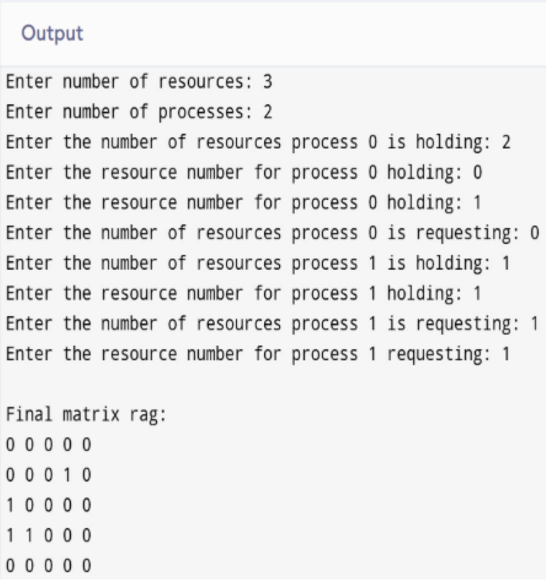
```
#include&lt;iostream&gt;
using namespace std;
int main()
{
int Resources[6][6],Process[5];
int resource=0,i,j,n,row,column,count=-1;
cout&lt;&lt;&quot;I have implemented a 4x3 matrix for resource allocation&quot;&lt;&lt;endl;
cout&lt;&lt;&quot;4 processes and 3 columns of resources&quot;&lt;&lt;endl;
cout&lt;&lt;&quot;Process Sequence&quot;&lt;&lt;endl;
cout&lt;&lt;&quot;Enter the number of processes&quot;&lt;&lt;endl;
cin&gt;&gt;n;
cout&lt;&lt;&quot;Enter Process Numbers (0-4)&quot;&lt;&lt;endl;
for(i=0;i&lt;n;i++)
{
cin&gt;&gt;Process[i];//Process count is entered here.Preferably 4-5 processes can be considered
}
cout&lt;&lt;&quot;Enter number of rows for resource array&quot;&lt;&lt;endl;
cin&gt;&gt;row; //Program is designed to handle 4 rows
cout&lt;&lt;&quot;Enter number of columns for resource array&quot;&lt;&lt;endl;
cin&gt;&gt;column; //Program is designed to handle 3 columns
cout&lt;&lt;&quot;Enter resources&quot;&lt;&lt;endl;
for(i=0;i&lt;row;i++)
{
for(j=0;j&lt;column;j++)
{
cin&gt;&gt;Resources[i][j]; //Resources are allocated in a i x j matrix
}
}
cout&lt;&lt;&quot;Process Show&quot;&lt;&lt;endl;

for(i=0;i&lt;n;i++)
{
cout&lt;&lt;&quot;P&quot;&lt;&lt;Process[i]&lt;&lt;endl; //To display processes
}
cout&lt;&lt;&quot;Resource Matrix Process Number&quot;&lt;&lt;endl;
cout&lt;&lt;&quot;A &quot;&lt;&lt;&quot; B &quot;&lt;&lt;&quot; C &quot;&lt;&lt;endl;
for(i=0;i&lt;1;i++)
{
for(j=0;j&lt;1;j++)
{
cout&lt;&lt;Resources[0][0]&lt;&lt;&quot; &quot;&lt;&lt;Resources[0][1]&lt;&lt;&quot;
&quot;&lt;&lt;Resources[0][2]&lt;&lt;&quot; &lt;-- &quot;&lt;&lt;Process[0]&lt;&lt;endl;
cout&lt;&lt;Resources[1][0]&lt;&lt;&quot; &quot;&lt;&lt;Resources[1][1]&lt;&lt;&quot;
&quot;&lt;&lt;Resources[1][2]&lt;&lt;&quot; &lt;-- &quot;&lt;&lt;Process[1]&lt;&lt;endl;
cout&lt;&lt;Resources[2][0]&lt;&lt;&quot; &quot;&lt;&lt;Resources[2][1]&lt;&lt;&quot;
&quot;&lt;&lt;Resources[2][2]&lt;&lt;&quot; &lt;-- &quot;&lt;&lt;Process[2]&lt;&lt;endl;
cout&lt;&lt;Resources[3][0]&lt;&lt;&quot; &quot;&lt;&lt;Resources[3][1]&lt;&lt;&quot;
&quot;&lt;&lt;Resources[3][2]&lt;&lt;&quot; &lt;-- &quot;&lt;&lt;Process[3]&lt;&lt;endl;
}
}
cout&lt;&lt;&quot;Check for Resource Competition/Possibility of Deadlock&quot;&lt;&lt;endl;
cout&lt;&lt;&quot;Please check resource matrix&quot;&lt;&lt;endl;
cout&lt;&lt;&quot;Enter a resource to check if collision happens&quot;&lt;&lt;endl;
cin&gt;&gt;resource;
for(i=0;i&lt;row;i++)
{
for(j=0;j&lt;column;j++)
{
if(resource==Resources[i][j])
{
cout&lt;&lt;&quot;Matches[row][column] &quot;&lt;&lt;i&lt;&lt;&quot;
&quot;&lt;&lt;j&lt;&lt;endl;
count=count+1;
}
}

}
cout&lt;&lt;&quot;Counts of deadlock = &quot;&lt;&lt;count&lt;&lt;endl;
return 0;
}
```



```
Enter number of resources: 3
Enter number of processes: 2
Enter the number of resources process 0 is holding: 2
Enter the resource number for process 0 holding: 0
Enter the resource number for process 0 holding: 1
Enter the number of resources process 0 is requesting: 0
Enter the number of resources process 1 is holding: 1
Enter the resource number for process 1 holding: 1
Enter the number of resources process 1 is requesting: 1
Enter the resource number for process 1 requesting: 1

Final matrix rag:
0 0 0 0 0
0 0 0 1 0
1 0 0 0 0
1 1 0 0 0
0 0 0 0 0
```

Q9 Process Scheduling

```
#include &lt;stdio.h&gt;
int main()
{

    int A[100][4];
    int i, j, n, total = 0, index, temp;
    float avg_wt, avg_tat;
    printf(&quot;Enter number of process: &quot;);
    scanf(&quot;%d&quot;, &amp;n);
    printf(&quot;Enter Burst Time:\n&quot;);
    // User Input Burst Time and alloting Process Id.
    for (i = 0; i &lt; n; i++) {
        printf(&quot;P%d: &quot;, i + 1);
        scanf(&quot;%d&quot;, &amp;A[i][1]);
        A[i][0] = i + 1;
    }
    // Sorting process according to their Burst Time.
    for (i = 0; i &lt; n; i++) {
        index = i;
        for (j = i + 1; j &lt; n; j++)
            if (A[j][1] &lt; A[index][1])
                index = j;
        temp = A[i][1];
        A[i][1] = A[index][1];
        A[index][1] = temp;
        temp = A[i][0];
        A[i][0] = A[index][0];
        A[index][0] = temp;
    }
    A[0][2] = 0;
    // Calculation of Waiting Times
    for (i = 1; i &lt; n; i++) {
        A[i][2] = 0;
        for (j = 0; j &lt; i; j++)
            A[i][2] += A[j][1];
```

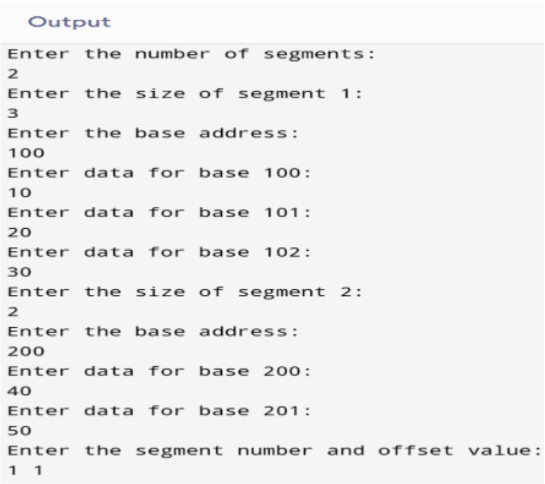


```
Enter number of resources: 3
Enter number of processes: 2
Enter the number of resources process 0 is holding: 2
Enter the resource number for process 0 holding: 0
Enter the resource number for process 0 holding: 1
Enter the number of resources process 0 is requesting: 0
Enter the number of resources process 1 is holding: 1
Enter the resource number for process 1 holding: 1
Enter the number of resources process 1 is requesting: 1
Enter the resource number for process 1 requesting: 1

Final matrix rag:
0 0 0 0 0
0 0 0 1 0
1 0 0 0 0
1 1 0 0 0
0 0 0 0 0
```

```
        total += A[i][2];
    }
    avg_wt = (float)total / n;
    total = 0;
    printf(&quot;P     BT     WT     TAT\n&quot;);
    // Calculation of Turn Around Time and printing the
    // data.
    for (i = 0; i &lt; n; i++) {
        A[i][3] = A[i][1] + A[i][2];
        total += A[i][3];
        printf(&quot;P%d     %d     %d      %d\n&quot;, A[i][0],
               A[i][1], A[i][2], A[i][3]);
    }    avg_tat = (float)total / n;
    printf(&quot;Average Waiting Time= %f&quot;, avg_wt);
    printf(&quot;\nAverage Turnaround Time= %f&quot;, avg_tat);}
```

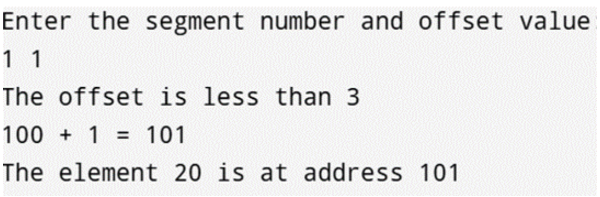
Q10 Segmentation

```
#include &lt;stdio.h&gt;
int main()
{
int n,nm,p,x=0,y=1,t=300,of,i;
printf(&quot;Enter the memory size:\n&quot;);
scanf(&quot;%d&quot;,&amp;nm);
printf(&quot;Enter the no.of segments:\n&quot;);
scanf(&quot;%d&quot;,&amp;n);
int s[n];
for(i=0;i&lt;n;i++)
{
printf(&quot;enter the segment size of %d:&quot;,i+1);
scanf(&quot;%d&quot;,&amp;s[i]);
x+=s[i];
if(x&gt;nm)
{
printf(&quot;memory full segment %d is not allocated&quot;,i+1);
x-=s[i];
s[i]=0;
}
}
printf(&quot;-----OPERATIONS------&quot;);
while(y==1)
{
printf(&quot;enter the no.of operations:\n&quot;);
scanf(&quot;%d&quot;,&amp;p);
printf(&quot;enter the offset:&quot;);
scanf(&quot;%d&quot;,&amp;of);
if(s[p-1]==0)

{
printf(&quot;segment is not allocated\n&quot;);
}
else if(of&gt;s[p-1])
{
printf(&quot;out of range!..&quot;);
}
else
{
printf(&quot;the segment %d the physical address is ranged from %d to %d\n the
address of operation
is\n&quot;,p,t,t+s[p-1],t+of);
}
printf(&quot;press 1 to continue&quot;);
scanf(&quot;%d&quot;,&amp;y);
}
}
```



Output

```
Enter the number of segments:
2
Enter the size of segment 1:
3
Enter the base address:
100
Enter data for base 100:
10
Enter data for base 101:
20
Enter data for base 102:
30
Enter the size of segment 2:
2
Enter the base address:
200
Enter data for base 200:
40
Enter data for base 201:
50
Enter the segment number and offset value:
1 1
```



```
Enter the segment number and offset value
1 1
The offset is less than 3
100 + 1 = 101
The element 20 is at address 101
```