Name: VENNELA G

Register No: 20BDS0146

Lab Course Name: OPERATING SYSTEMS

Lab Slot: L21+L22

Assesment No : 5

Question 1

Write a C program to provide a solution to the classical synchronization problem namely the Producer Consumer problem using POSIX Semaphores.

Answer:

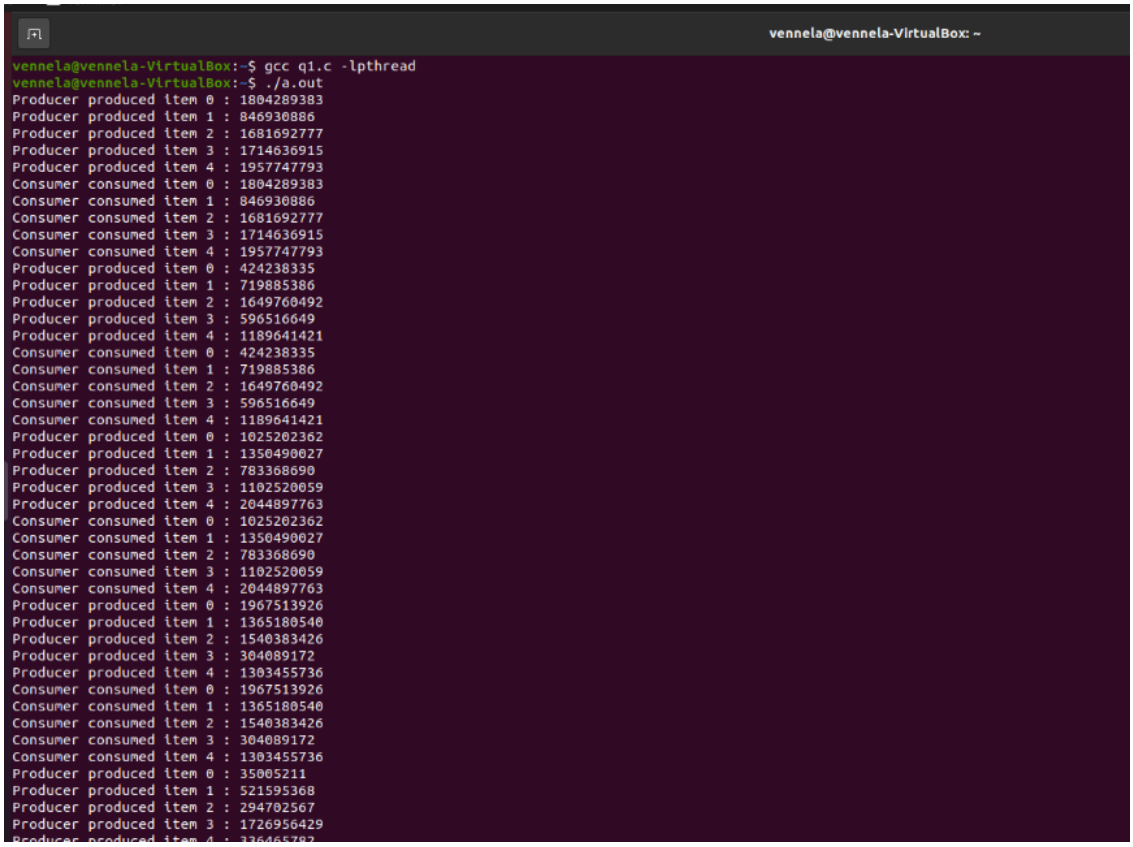
SOURCE CODE:

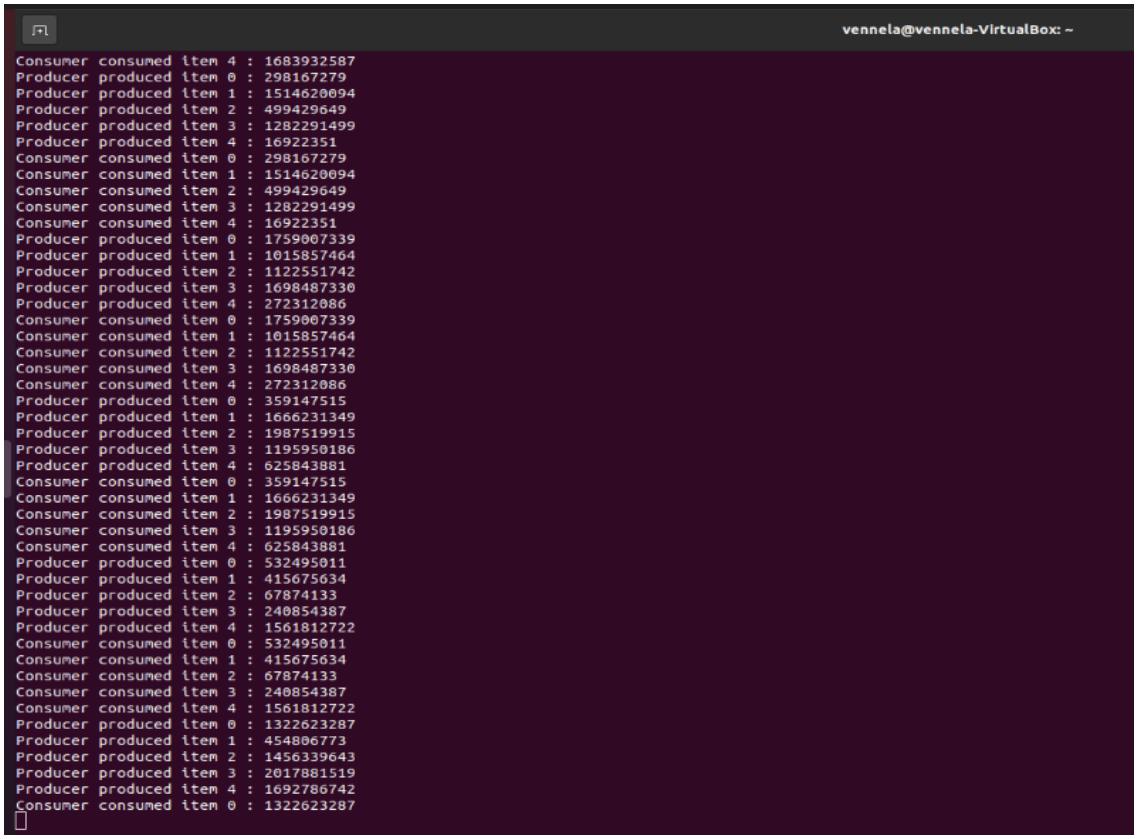
```
#include<stdio.h>
#include<pthread.h>
#include<sys/types.h>
#include<unistd.h>
#include<semaphore.h>
#include<stdlib.h>


#define BUFFER_SIZE 5


sem_t mutex, full, empty ;


int buffer[BUFFER_SIZE];
int in=0, out=0;
void * producer(void * arg) {
int nextProduced;
while ( 1 ) {


nextProduced = rand();
sem_wait( &empty );
sem_wait( &mutex );


buffer[in] = nextProduced;
printf("Producer produced item %d : %d \n", in, nextProduced);
```

```
usleep(100);
in = (in+1) % BUFFER_SIZE;




sem_post( &mutex );
sem_post( &full );
}



}



void * consumer(void * arg) {
int nextConsumed;
while ( 1 ) {



sem_wait( &full );
sem_wait( &mutex );



nextConsumed= buffer[out] ;
printf("Consumer consumed item %d : %d \n", out, nextConsumed);
usleep(100) ;
out = (out+1) % BUFFER_SIZE;




sem_post( &mutex );
sem_post( &empty );
}



}




int main() {
pthread_t pTID, cTID;
```

```
sem_init( &mutex, 0 , 1);
sem_init( &full, 0 , 0);
sem_init( &empty, 0 , BUFFER_SIZE);



if( pthread_create( &pTID, NULL, producer, NULL) < 0 ) {
perror("pthread_create");
exit( -1 );
}

if( pthread_create( &cTID, NULL, consumer, NULL) < 0 ) {
perror("pthread_create");
exit( -1 );
}

pthread_join(pTID, NULL);
pthread_join(cTID, NULL);
return 0;
}
```

OUTPUT:




vennela@vennela-VirtualBox: ~
Consumer consumed item 4 : 1683932587
Producer produced item 0 : 298167279
Producer produced item 1 : 1514620094
Producer produced item 2 : 499429649
Producer produced item 3 : 1282291499
Producer produced item 4 : 16922351
Consumer consumed item 0 : 298167279
Consumer consumed item 1 : 1514620094
Consumer consumed item 2 : 499429649
Consumer consumed item 3 : 1282291499
Consumer consumed item 4 : 16922351
Producer produced item 0 : 1759007339
Producer produced item 1 : 1015857464
Producer produced item 2 : 1122551742
Producer produced item 3 : 1698487330
Producer produced item 4 : 272312086
Consumer consumed item 0 : 1759007339
Consumer consumed item 1 : 1015857464
Consumer consumed item 2 : 1122551742
Consumer consumed item 3 : 1698487330
Consumer consumed item 4 : 272312086
Producer produced item 0 : 359147515
Producer produced item 1 : 1666231349
Producer produced item 2 : 1987519915
Producer produced item 3 : 1195950186
Producer produced item 4 : 625843881
Consumer consumed item 0 : 359147515
Consumer consumed item 1 : 1666231349
Consumer consumed item 2 : 1987519915
Consumer consumed item 3 : 1195950186
Consumer consumed item 4 : 625843881
Producer produced item 0 : 532495011
Producer produced item 1 : 415675634
Producer produced item 2 : 67874133
Producer produced item 3 : 240854387
Producer produced item 4 : 1561812722
Consumer consumed item 0 : 532495011
Consumer consumed item 1 : 415675634
Consumer consumed item 2 : 67874133
Consumer consumed item 3 : 240854387
Consumer consumed item 4 : 1561812722
Producer produced item 0 : 1322623287
Producer produced item 1 : 454806773
Producer produced item 2 : 1456339643
Producer produced item 3 : 2017881519
Producer produced item 4 : 1692786742
Consumer consumed item 0 : 1322623287

Question 2

Write a C program to study the working of Readers Writer problem using POSIX Semaphores.

Answer:

SOURCE CODE:

Writer process:

```
#include<unistd.h>
#include<stdio.h>
#include<sys/types.h>
#include<sys/mman.h>
#include<fcntl.h>
#include<stdlib.h>
#include<string.h>
char msg[100];
int main()
{int shmFD,s;char mod[100];
char * addr;
shmFD=shm_open("Shm1",O_RDWR|O_CREAT,0666);
if(shmFD==-1)
{perror("shm_open");
exit(-1);}
if(ftruncate(shmFD,512)==-1)
{perror("ftruncate");
exit(-1);}
addr=mmap(NULL,512,PROT_WRITE,MAP_SHARED,shmFD,0);
```

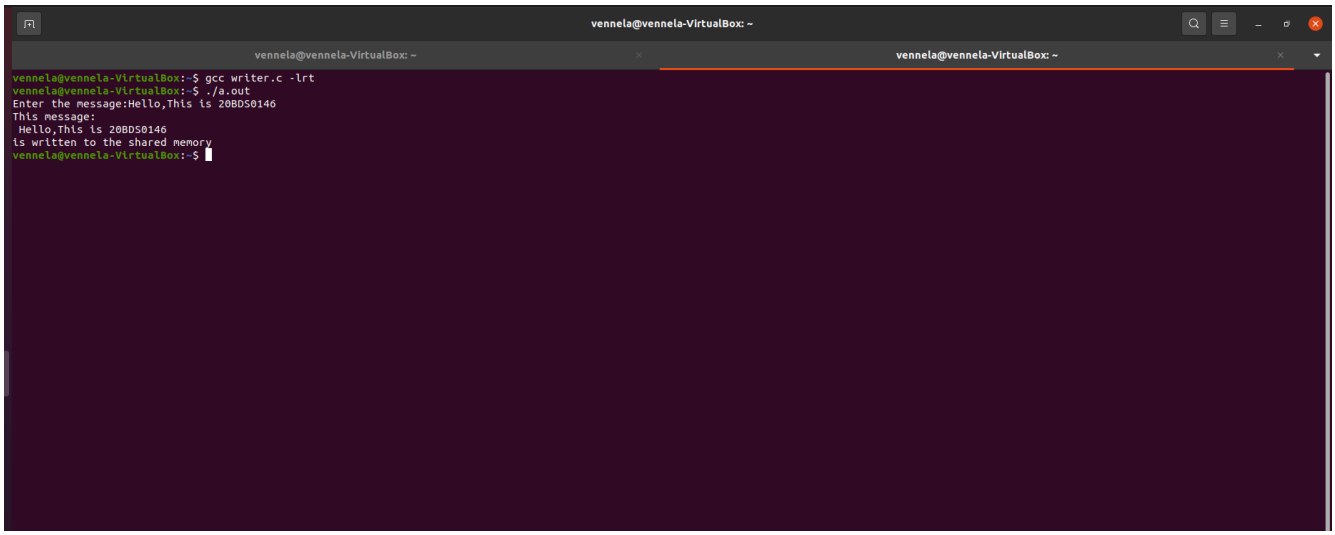
```
if(addr==MAP_FAILED)
{perror("mmap");
exit(-1);}
printf("Enter the message:");
scanf("%[^\n]s",msg);
memcpy(addr,msg,strlen(msg)+1);
printf("This message:\n %s \nis written to the shared memory\n",msg);

return 0;
}
```

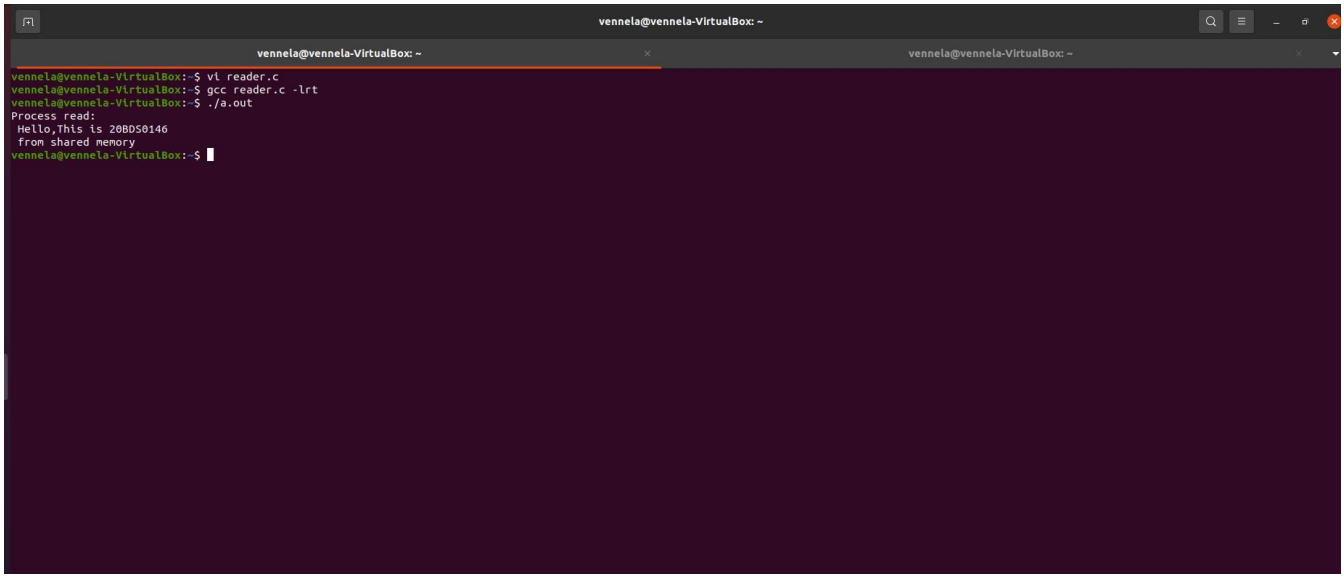
Reader process:

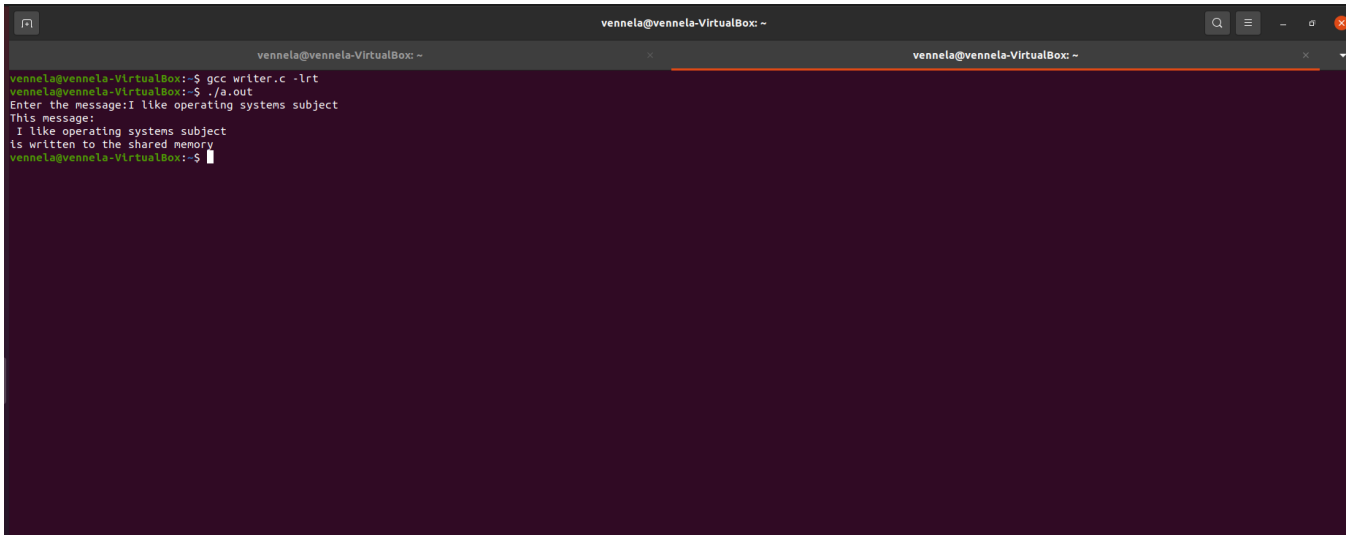
```
#include<stdio.h>
#include<sys/types.h>
#include<sys/mman.h>
#include<fcntl.h>
#include<stdlib.h>
#include<string.h>
int main()
{int shmFD;
void *addr;
char msg[100];
shmFD=shm_open("Shm1",O_RDONLY,0666);
if(shmFD==-1){perror("shm_open");
exit(-1);}
addr=mmap(NULL,512,PROT_READ,MAP_SHARED,shmFD,0);
```

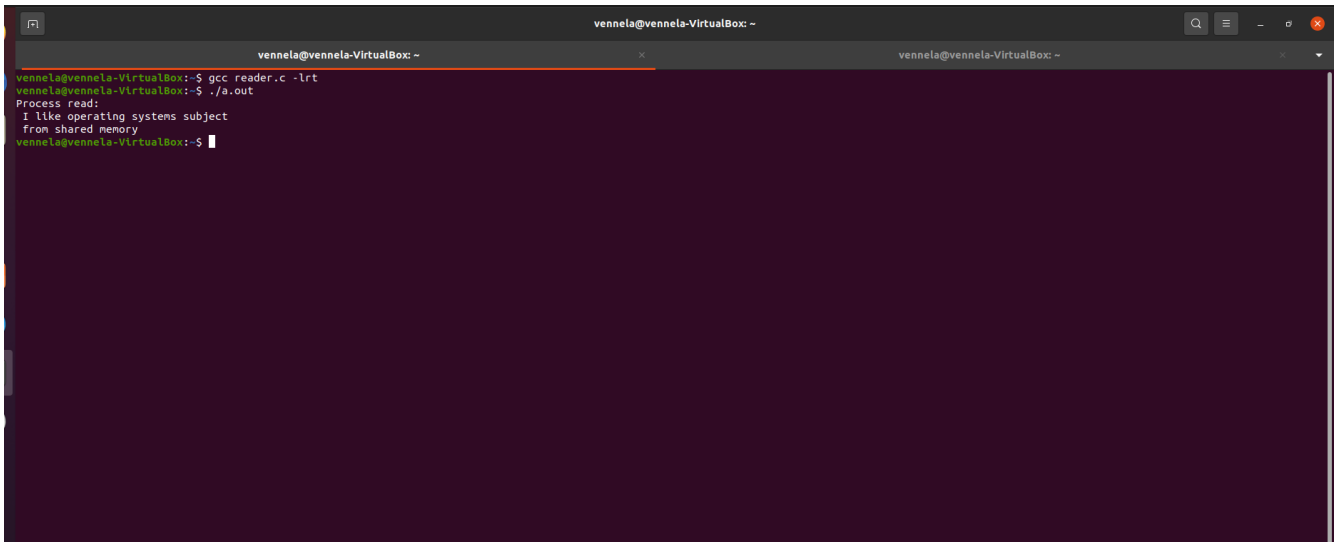
```
if(addr==MAP_FAILED)
{perror("mmap");
exit(-1);}
memcpy(msg,addr,sizeof(msg));
printf("Process read:\n %s \n from shared memory\n",msg);
return 0;
}
```

OUTPUT:






vennela@vennela-VirtualBox: ~
vennela@vennela-VirtualBox: ~
vennela@vennela-VirtualBox: ~
vennela@vennela-VirtualBox:~$ gcc writer.c -lrt
vennela@vennela-VirtualBox:~$ ./a.out
Enter the message:I like operating systems subject
This message:
I like operating systems subject
is written to the shared memory
vennela@vennela-VirtualBox:~$


vennela@vennela-VirtualBox: ~
vennela@vennela-VirtualBox: ~
vennela@vennela-VirtualBox: ~
vennela@vennela-VirtualBox:~$ gcc reader.c -lrt
vennela@vennela-VirtualBox:~$ ./a.out
Process read:
I like operating systems subject
from shared memory
vennela@vennela-VirtualBox:~$

Question 3

Write a C program to study the working of Dining Philosophers using POSIX Semaphores

Answer:

SOURCE CODE:

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

#define N 5
#define THINKING 2
#define HUNGRY 1
#define EATING 0
#define LEFT (i + 4) % N
#define RIGHT (i + 1) % N

int state[N];
int phil[N] = { 0, 1, 2, 3, 4 };

sem_t mutex;
sem_t S[N];

void test(int i)
{
        if (state[i] == HUNGRY
                && state[LEFT] != EATING
```

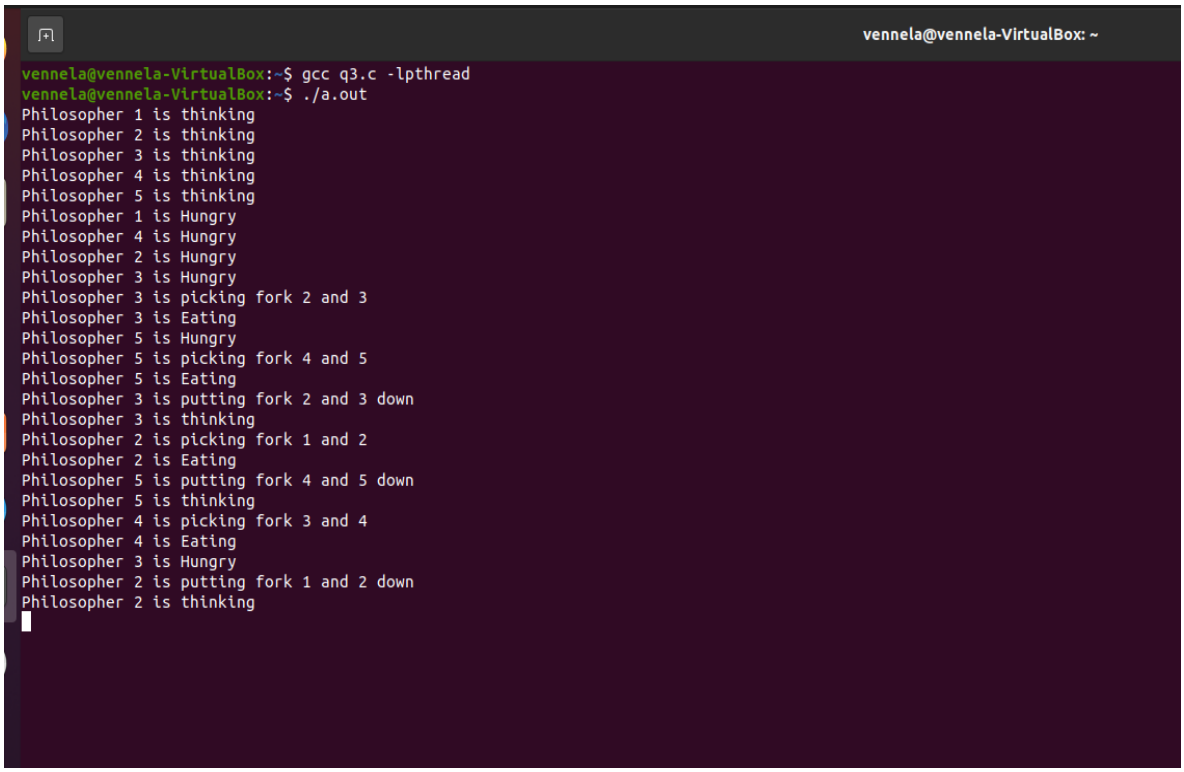
```
                && state[RIGHT] != EATING) {

                state[i] = EATING;

                sleep(3);

                printf("Philosopher %d is picking fork %d and %d\n",i + 1, LEFT + 1, i + 1);

                printf("Philosopher %d is Eating\n", i + 1);
                sem_post(&S[i]);
        }
}


void pick_up(int i)
{
        sem_wait(&mutex);


        state[i] = HUNGRY;

        printf("Philosopher %d is Hungry\n", i+ 1);


        test(i);

        sem_post(&mutex);
```

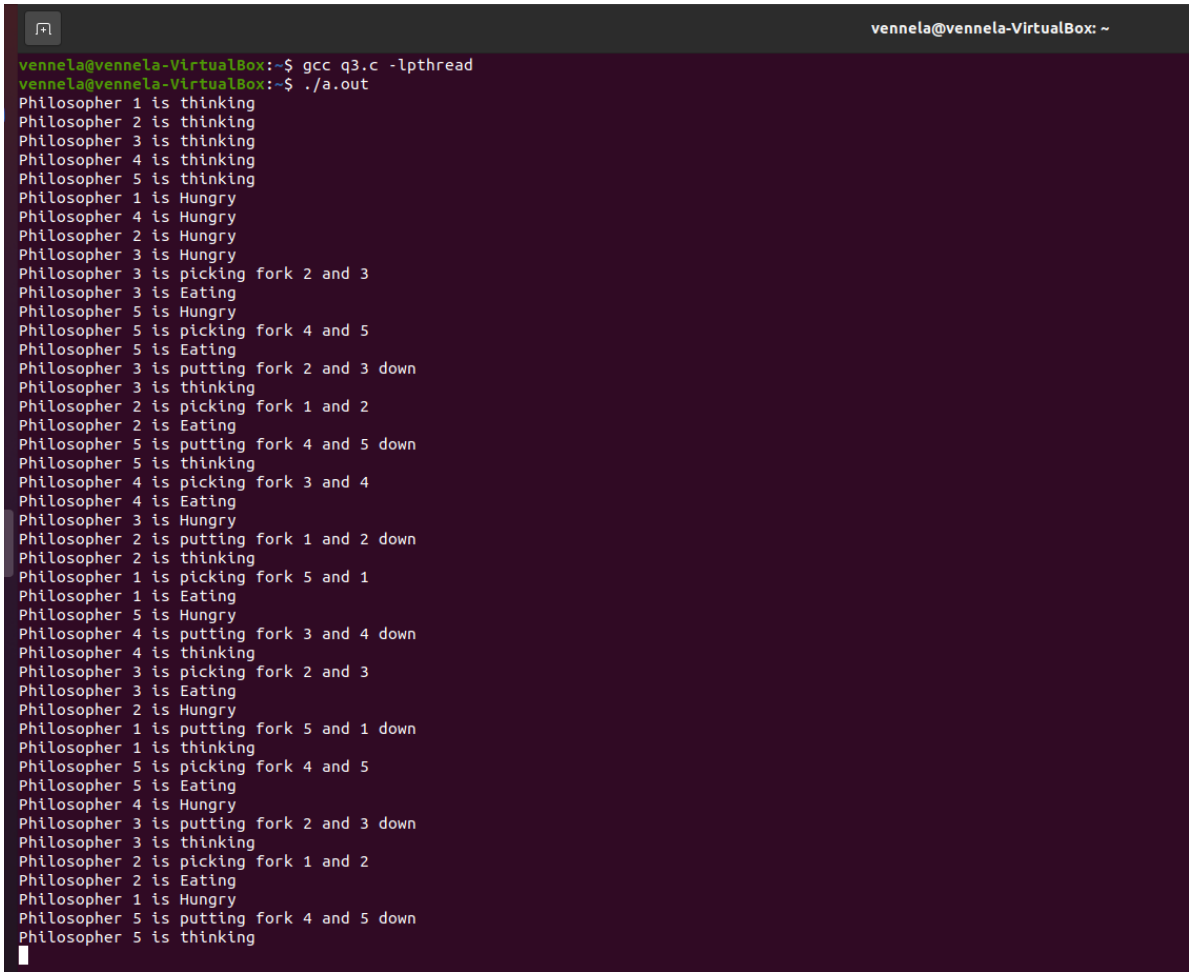
```
        sem_wait(&S[i]);

        sleep(1);
}


void put_down(int i)
{

        sem_wait(&mutex);


        state[i] = THINKING;

        printf("Philosopher %d is putting fork %d and %d down\n",
                i + 1, LEFT + 1, i + 1);
        printf("Philosopher %d is thinking\n", i+ 1);

        test(LEFT);
        test(RIGHT);

        sem_post(&mutex);
}

void* philosopher(void* num)
{

        while (1) {
```

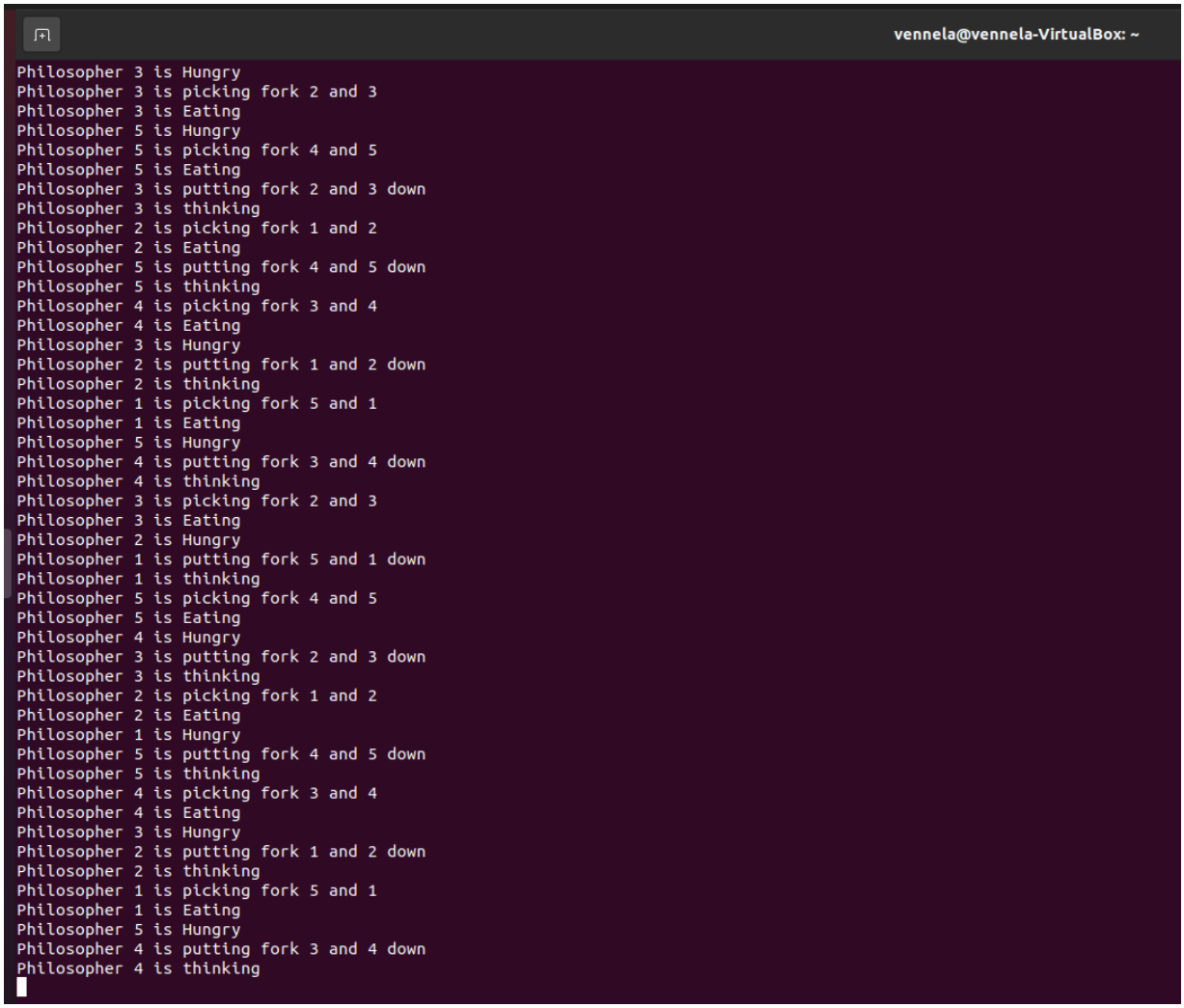
```
            int* i = num;

            sleep(1);

            pick_up(*i);

            sleep(0);

            put_down(*i);
        }
}

int main()
{

        int i;
        pthread_t thread_id[N];

        sem_init(&mutex, 0, 1);

        for (i = 0; i < N; i++)

            sem_init(&S[i], 0, 0);

        for (i = 0; i < N; i++) {

            pthread_create(&thread_id[i], NULL,
```

```
                philosopher, &phil[i]);

            printf("Philosopher %d is thinking\n", i + 1);
        }

        for (i = 0; i < N; i++)

            pthread_join(thread_id[i], NULL);

            return 0;
}
```

OUTPUT:




vennela@vennela-VirtualBox: ~
vennela@vennela-VirtualBox:~$ gcc q3.c -lpthread
vennela@vennela-VirtualBox:~$ ./a.out
Philosopher 1 is thinking
Philosopher 2 is thinking
Philosopher 3 is thinking
Philosopher 4 is thinking
Philosopher 5 is thinking
Philosopher 1 is Hungry
Philosopher 4 is Hungry
Philosopher 2 is Hungry
Philosopher 3 is Hungry
Philosopher 3 is picking fork 2 and 3
Philosopher 3 is Eating
Philosopher 5 is Hungry
Philosopher 5 is picking fork 4 and 5
Philosopher 5 is Eating
Philosopher 3 is putting fork 2 and 3 down
Philosopher 3 is thinking
Philosopher 2 is picking fork 1 and 2
Philosopher 2 is Eating
Philosopher 5 is putting fork 4 and 5 down
Philosopher 5 is thinking
Philosopher 4 is picking fork 3 and 4
Philosopher 4 is Eating
Philosopher 3 is Hungry
Philosopher 2 is putting fork 1 and 2 down
Philosopher 2 is thinking
Philosopher 1 is picking fork 5 and 1
Philosopher 1 is Eating
Philosopher 5 is Hungry
Philosopher 4 is putting fork 3 and 4 down
Philosopher 4 is thinking
Philosopher 3 is picking fork 2 and 3
Philosopher 3 is Eating
Philosopher 2 is Hungry
Philosopher 1 is putting fork 5 and 1 down
Philosopher 1 is thinking
Philosopher 5 is picking fork 4 and 5
Philosopher 5 is Eating
Philosopher 4 is Hungry
Philosopher 3 is putting fork 2 and 3 down
Philosopher 3 is thinking
Philosopher 2 is picking fork 1 and 2
Philosopher 2 is Eating
Philosopher 1 is Hungry
Philosopher 5 is putting fork 4 and 5 down
Philosopher 5 is thinking


vennela@vennela-VirtualBox: ~
Philosopher 3 is Hungry
Philosopher 3 is picking fork 2 and 3
Philosopher 3 is Eating
Philosopher 5 is Hungry
Philosopher 5 is picking fork 4 and 5
Philosopher 5 is Eating
Philosopher 3 is putting fork 2 and 3 down
Philosopher 3 is thinking
Philosopher 2 is picking fork 1 and 2
Philosopher 2 is Eating
Philosopher 5 is putting fork 4 and 5 down
Philosopher 5 is thinking
Philosopher 4 is picking fork 3 and 4
Philosopher 4 is Eating
Philosopher 3 is Hungry
Philosopher 2 is putting fork 1 and 2 down
Philosopher 2 is thinking
Philosopher 1 is picking fork 5 and 1
Philosopher 1 is Eating
Philosopher 5 is Hungry
Philosopher 4 is putting fork 3 and 4 down
Philosopher 4 is thinking
Philosopher 3 is picking fork 2 and 3
Philosopher 3 is Eating
Philosopher 2 is Hungry
Philosopher 1 is putting fork 5 and 1 down
Philosopher 1 is thinking
Philosopher 5 is picking fork 4 and 5
Philosopher 5 is Eating
Philosopher 4 is Hungry
Philosopher 3 is putting fork 2 and 3 down
Philosopher 3 is thinking
Philosopher 2 is picking fork 1 and 2
Philosopher 2 is Eating
Philosopher 1 is Hungry
Philosopher 5 is putting fork 4 and 5 down
Philosopher 5 is thinking
Philosopher 4 is picking fork 3 and 4
Philosopher 4 is Eating
Philosopher 3 is Hungry
Philosopher 2 is putting fork 1 and 2 down
Philosopher 2 is thinking
Philosopher 1 is picking fork 5 and 1
Philosopher 1 is Eating
Philosopher 5 is Hungry
Philosopher 4 is putting fork 3 and 4 down
Philosopher 4 is thinking

Question 4

Code the Banker's algorithm in C and test the working of it with arbitrary inputs.

Answer:

SOURCE CODE:

```
#include <stdio.h>

int current[5][5], maximum_claim[5][5], available[5];
int allocation[5] = {0, 0, 0, 0, 0};
int maxres[5], running[5], safe = 0;
int count = 0, i, j, exec, res, proc, k = 1;

int main()
{
printf("\nEnter number of processes: ");
    scanf("%d", &proc);

    for (i = 0; i < proc; i++)
{
        running[i] = 1;
        count++;
    }

    printf("\nEnter number of resources: ");
    scanf("%d", &res);
```

```
    printf("\nEnter instances of resources Vector:");
    for (i = 0; i < res; i++)
{
       scanf("%d", &maxres[i]);
    }

   printf("\nEnter Allocated Resource Table:\n");
    for (i = 0; i < proc; i++)
{
       for(j = 0; j < res; j++)
{
   scanf("%d", &current[i][j]);
        }
    }

    printf("\nEnter Maximum Claim Table:\n");
    for (i = 0; i < proc; i++)
{
        for(j = 0; j < res; j++)
{
           scanf("%d", &maximum_claim[i][j]);
        }
    }

printf("\nThe Claim Vector is: ");
    for (i = 0; i < res; i++)
{
       printf("\t%d", maxres[i]);
```

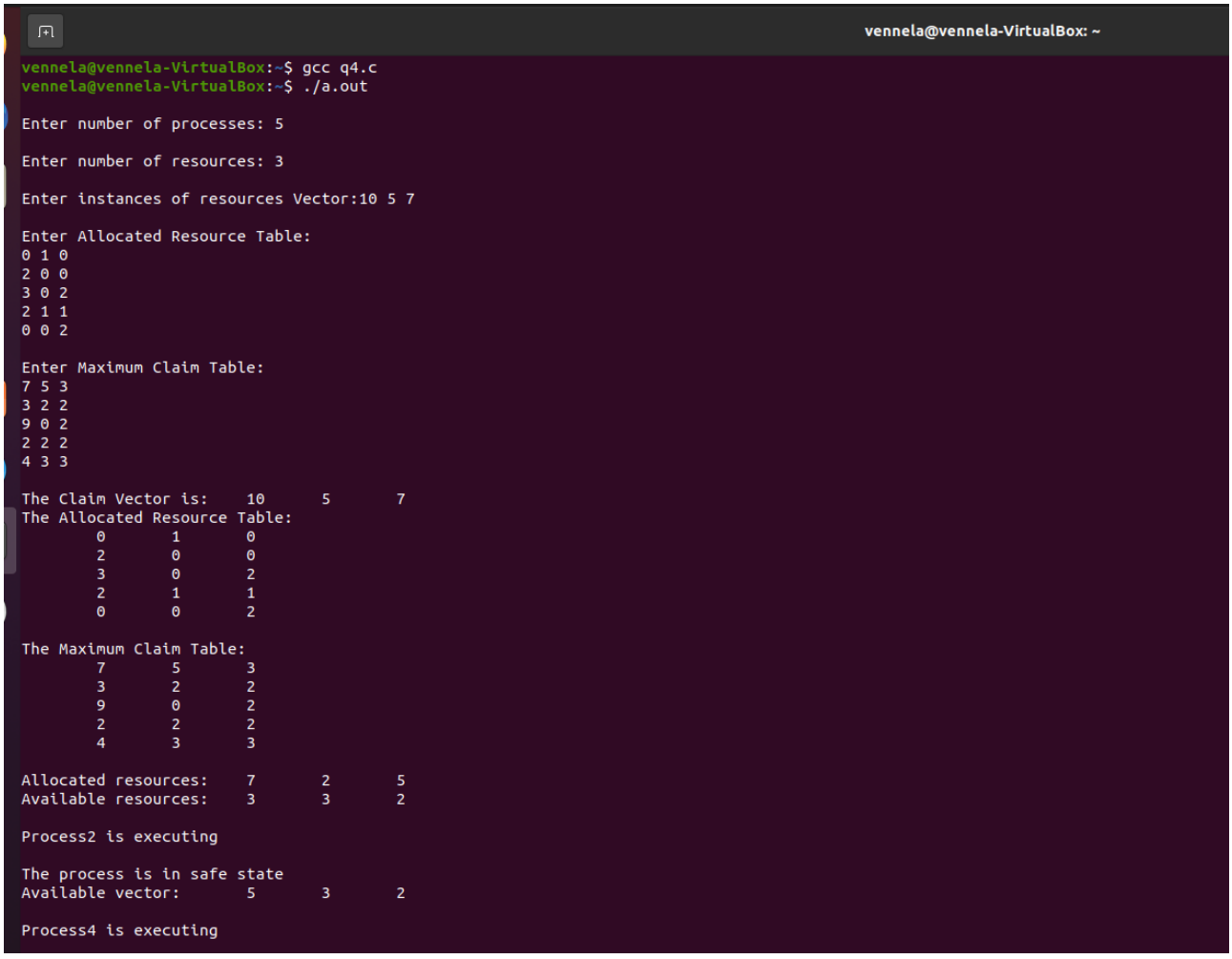
```
}

    printf("\nThe Allocated Resource Table:\n");
    for (i = 0; i < proc; i++)
{
      for (j = 0; j < res; j++)
{
          printf("\t%d", current[i][j]);
       }
printf("\n");
    }

    printf("\nThe Maximum Claim Table:\n");
    for (i = 0; i < proc; i++)
{
       for (j = 0; j < res; j++)
{
      printf("\t%d", maximum_claim[i][j]);
       }
       printf("\n");
    }

    for (i = 0; i < proc; i++)
{
       for (j = 0; j < res; j++)
{
          allocation[j] += current[i][j];
       }
    }
```

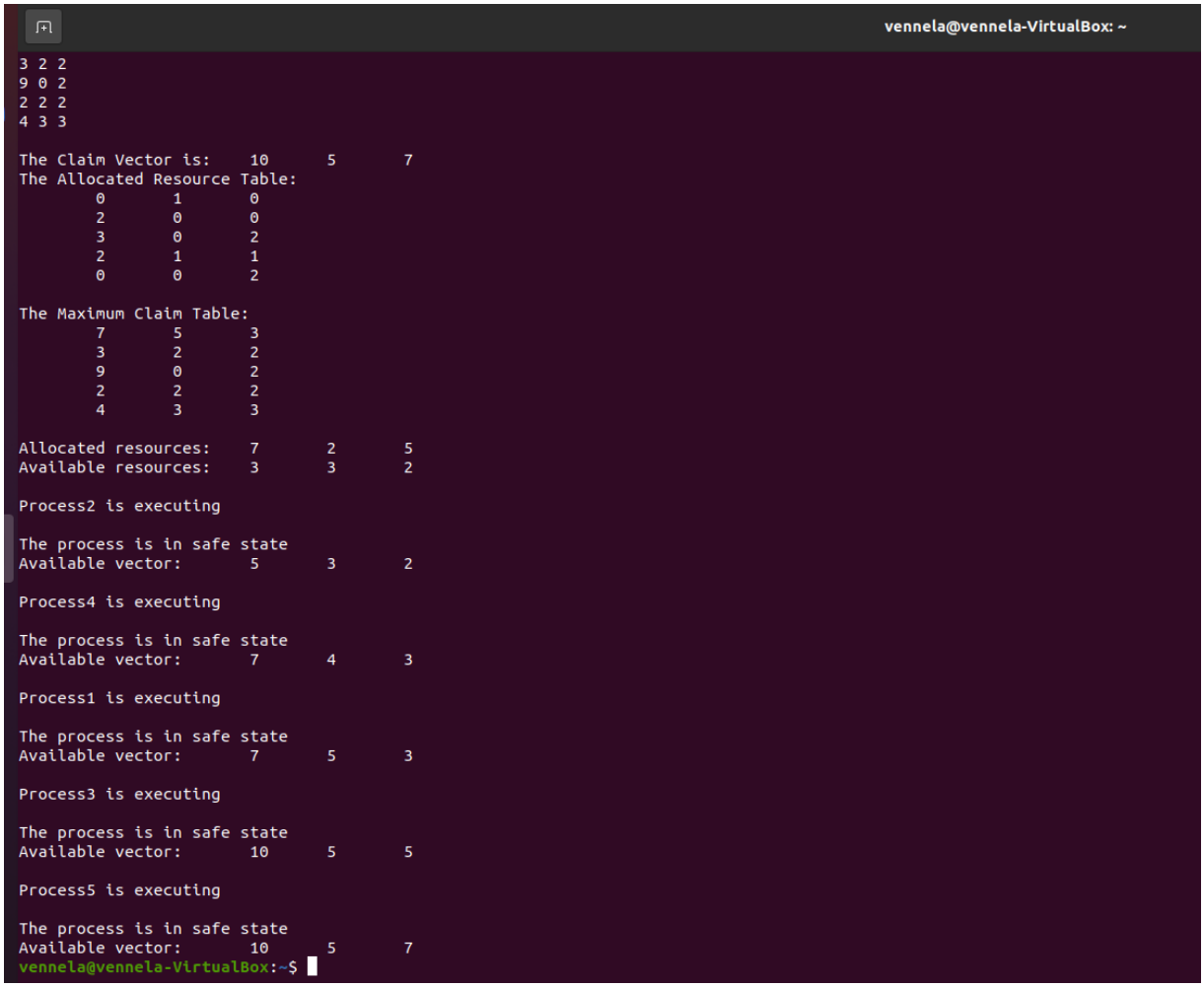
```
    printf("\nAllocated resources:");
    for (i = 0; i < res; i++)
{
        printf("\t%d", allocation[i]);
    }

    for (i = 0; i < res; i++)
{
       available[i] = maxres[i] - allocation[i];
}

    printf("\nAvailable resources:");
    for (i = 0; i < res; i++)
{
        printf("\t%d", available[i]);
    }
    printf("\n");

    while (count!= 0)
{
        safe = 0;
        for (i = 0; i < proc; i++)
{
            if (running[i])
{
                exec = 1;
                for (j = 0; j < res; j++)
{
```

```
                if (maximum_claim[i][j] - current[i][j] > available[j])
{
                    exec = 0;
                    break;
                }
            }
            if (exec)
{
                printf("\nProcess%d is executing\n", i + 1);
                running[i] = 0;
                count--;
                safe = 1;

                for (j = 0; j < res; j++)
{
                    available[j] += current[i][j];
                }
            break;
             }
          }
       }
       if (!safe)
{
          printf("\nThe processes are in unsafe state.\n");
          break;
       }
else
{
          printf("\nThe process is in safe state");
```

```
        printf("\nAvailable vector:");

        for (i = 0; i < res; i++)
{
            printf("\t%d", available[i]);
        }

      printf("\n");
       }
   }
   return 0;
}
```

OUTPUT:




vennela@vennela-VirtualBox: ~
3 2 2
9 0 2
2 2 2
4 3 3
The Claim Vector is: 10 5 7
The Allocated Resource Table:
0 1 0
2 0 0
3 0 2
2 1 1
0 0 2
The Maximum Claim Table:
7 5 3
3 2 2
9 0 2
2 2 2
4 3 3
Allocated resources: 7 2 5
Available resources: 3 3 2
Process2 is executing
The process is in safe state
Available vector: 5 3 2
Process4 is executing
The process is in safe state
Available vector: 7 4 3
Process1 is executing
The process is in safe state
Available vector: 7 5 3
Process3 is executing
The process is in safe state
Available vector: 10 5 5
Process5 is executing
The process is in safe state
Available vector: 10 5 7
vennela@vennela-VirtualBox:~$

```
vennela@vennela-VirtualBox: ~
vennela@vennela-VirtualBox:~$ vi q4.c
vennela@vennela-VirtualBox:~$ gcc q4.c
vennela@vennela-VirtualBox:~$ ./a.out

Enter number of processes: 5

Enter number of resources: 4

Enter instances of resources Vector:3 17 16 12

Enter Allocated Resource Table:
0 1 1 0
1 2 3 1
1 3 6 5
0 6 3 2
0 0 1 4

Enter Maximum Claim Table:
0 2 1 0
1 6 5 2
2 3 6 6
0 6 5 2
0 6 5 6

The Claim Vector is:    3       17      16      12
The Allocated Resource Table:
        0       1       1       0
        1       2       3       1
        1       3       6       5
        0       6       3       2
        0       0       1       4

The Maximum Claim Table:
        0       2       1       0
        1       6       5       2
        2       3       6       6
        0       6       5       2
        0       6       5       6

Allocated resources:    2       12      14      12
Available resources:    1       5       2       0

Process1 is executing

The process is in safe state
Available vector:       1       6       3       0

Process4 is executing
```

```
vennela@vennela-VirtualBox: ~
1 6 5 2
2 3 6 6
0 6 5 2
0 6 5 6

The Claim Vector is:    3       17      16      12
The Allocated Resource Table:
        0       1       1       0
        1       2       3       1
        1       3       6       5
        0       6       3       2
        0       0       1       4

The Maximum Claim Table:
        0       2       1       0
        1       6       5       2
        2       3       6       6
        0       6       5       2
        0       6       5       6

Allocated resources:    2       12      14      12
Available resources:    1       5       2       0

Process1 is executing

The process is in safe state
Available vector:       1       6       3       0

Process4 is executing

The process is in safe state
Available vector:       1       12      6       2

Process2 is executing

The process is in safe state
Available vector:       2       14      9       3

Process3 is executing

The process is in safe state
Available vector:       3       17      15      8

Process5 is executing

The process is in safe state
Available vector:       3       17      16      12
vennela@vennela-VirtualBox:~$
```

Question 5

Write a C program to study the allocation of memory by applying the following memory allocation strategies.

- FIRST FIT
- BEST FIT
- WORST FIT

Answer:

SOURCE CODE:

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

void firstFit()
{ int blockSize[20],processSize[20];
int a1,b1;
printf("Enter number of blocks: ");
scanf("%d", &a1);
printf("\nEnter the size of each block:\n ");
for (int i = 0; i < a1; i++)
{
printf("Block no.%d: ", i);
scanf("%d", &blockSize[i]);
}
printf("\nEnter no. of processes: ");
scanf("%d",&b1);
```

```
printf("\nEnter size of each process:\n ");
for (int i = 0; i < b1; i++)
{
printf("Process no.%d: ", i);
scanf("%d", &processSize[i]);
}


        printf("\n\t\t\tMemory Management Scheme-First Fit\n");
        int allocation[b1];


        memset(allocation, -1, sizeof(allocation));


        for (int i = 0; i < b1; i++)
        {
                for (int j = 0; j < a1; j++)
                {
                        if (blockSize[j] >= processSize[i])
                        {

                                allocation[i] = j;


                                blockSize[j] -= processSize[i];

                                break;
                        }
```

```
                }
        }

        printf("\nProcess No.\tProcess Size\tBlock no.\n");
        for (int i = 0; i < b1; i++)
        {
                printf("%d\t\t%d\t\t",i+1,processSize[i]);

                if (allocation[i] != -1)
                        printf("%d", allocation[i] + 1);
                else
                        printf( "Not Allocated");
                printf("\n");
        }
}
void bestFit()
{  int blockSize[20],processSize[20];
int a1,b1;
printf("Enter number of blocks: ");
scanf("%d", &a1);
printf("\nEnter the size of each block:\n ");
for (int i = 0; i < a1; i++)
{
printf("Block no.%d: ", i);
scanf("%d", &blockSize[i]);
}
printf("\nEnter no. of processes: ");
scanf("%d",&b1);
```

```
printf("\nEnter size of each process:\n ");
for (int i = 0; i < b1; i++)
{
printf("Process no.%d: ", i);
scanf("%d", &processSize[i]);
}


   printf("\n\t\t\tMemory Management Scheme-Best Fit\n");
        int allocation1[b1];

        memset(allocation1, -1, sizeof(allocation1));

        for (int i=0; i<b1; i++)
        {

                int bestIdx = -1;
                for (int j=0; j<a1; j++)
                {
                        if (blockSize[j] >= processSize[i])
                        {
                                if (bestIdx == -1)
                                        bestIdx = j;
                                else if (blockSize[bestIdx] > blockSize[j])
                                        bestIdx = j;
                        }
                }

                // If we could find a block for current process
```

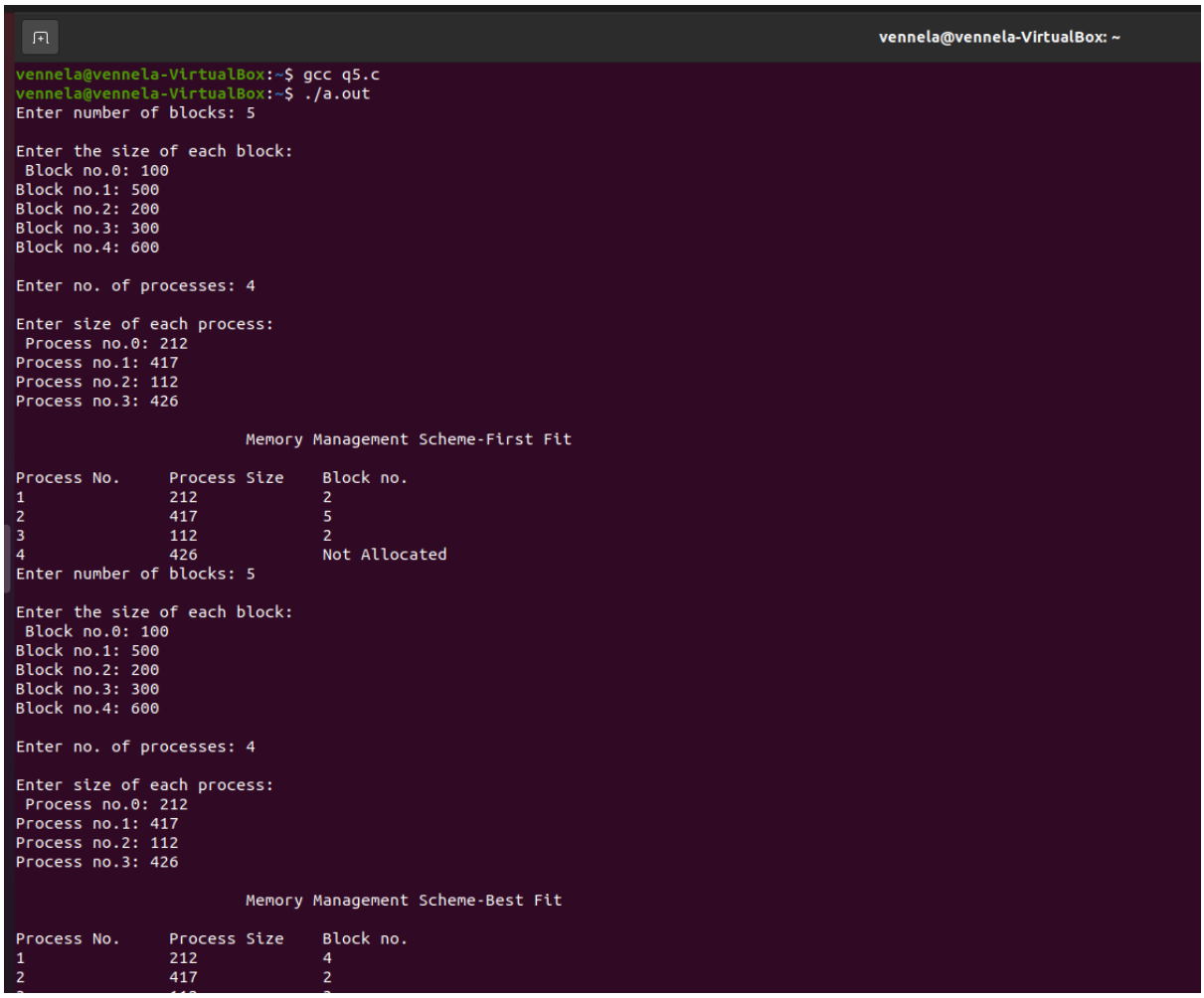
```
                if (bestIdx != -1)
                {
                        // allocate block j to p[i] process
                        allocation1[i] = bestIdx;

                        // Reduce available memory in this block.
                        blockSize[bestIdx] -= processSize[i];
                }
        }

        printf("\nProcess No.\tProcess Size\tBlock no.\n");
        for (int i = 0; i < b1; i++)
        {
                printf("%d\t\t%d\t\t",i+1,processSize[i]);
                if (allocation1[i] != -1)
                        printf("%d",allocation1[i] + 1) ;
                else
                   printf("Not Allocated") ;
           printf("\n");

        }
}
void worstFit()
{   int blockSize[20],processSize[20];
int a1,b1;
printf("Enter number of blocks: ");
scanf("%d", &a1);
printf("\nEnter the size of each block:\n ");
for (int i = 0; i < a1; i++)
```

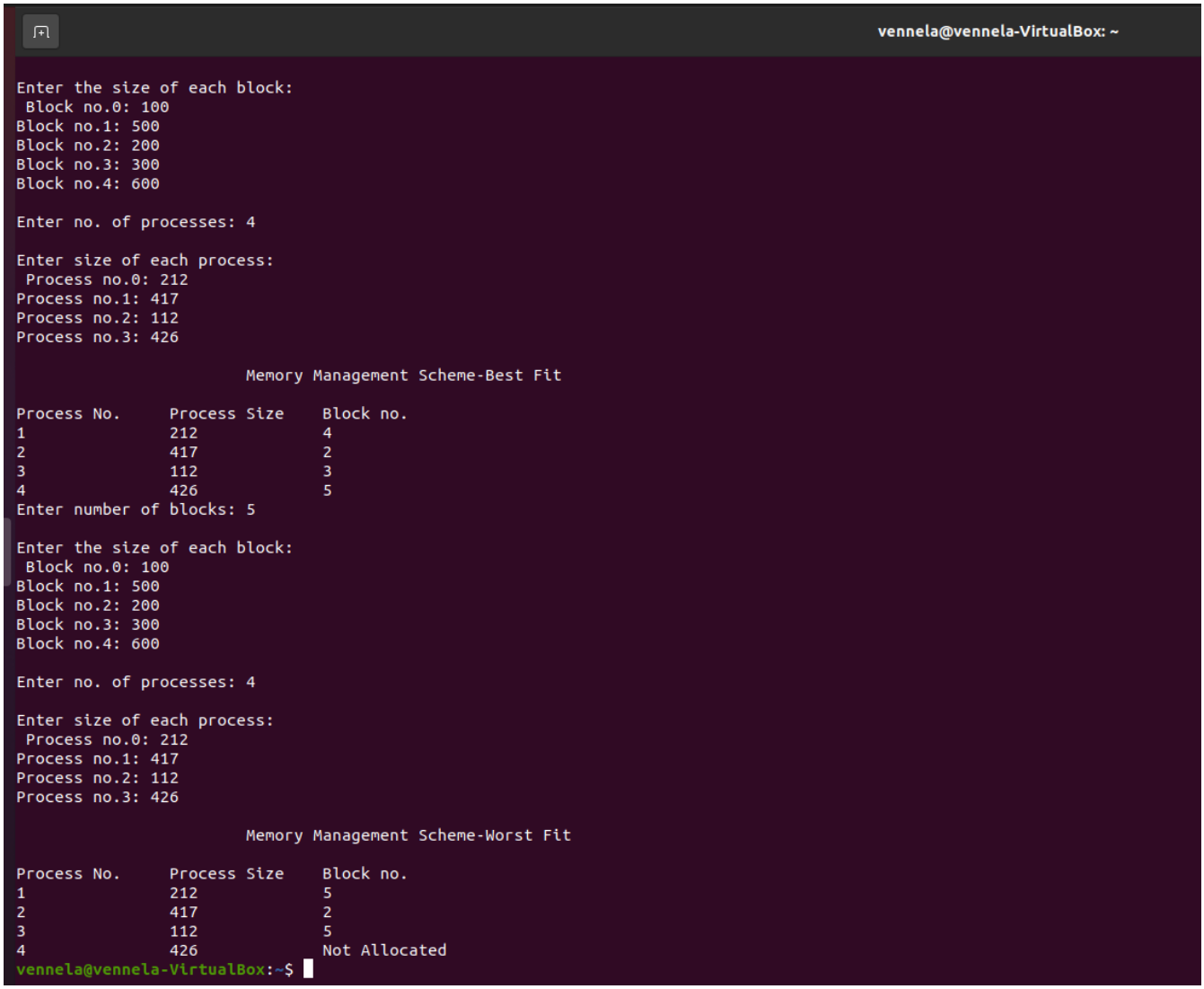
```
{
printf("Block no.%d: ", i);
scanf("%d", &blockSize[i]);
}
printf("\nEnter no. of processes: ");
scanf("%d",&b1);

printf("\nEnter size of each process:\n ");
for (int i = 0; i < b1; i++)
{
printf("Process no.%d: ", i);
scanf("%d", &processSize[i]);
}


        printf("\n\t\t\tMemory Management Scheme-Worst Fit\n");
        int allocation2[b1];

        memset(allocation2, -1, sizeof(allocation2));


        for (int i=0; i<b1; i++)
        {

                int wstIdx = -1;
                for (int j=0; j<a1; j++)
                {
                        if (blockSize[j] >= processSize[i])
                        {
```

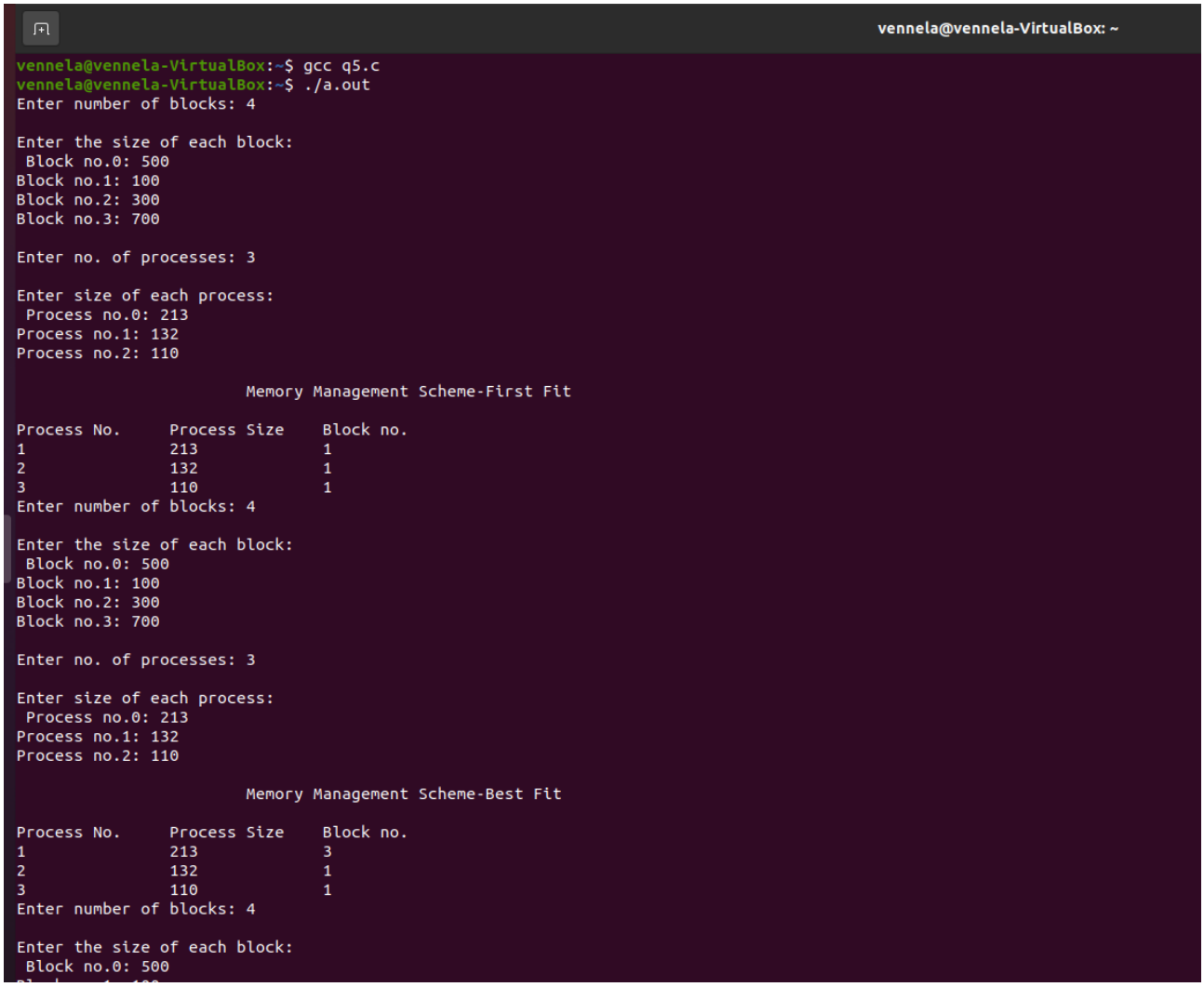
```
                        if (wstIdx == -1)
                                wstIdx = j;
                        else if (blockSize[wstIdx] < blockSize[j])
                                wstIdx = j;
                    }
                }


                if (wstIdx != -1)
                {

                    allocation2[i] = wstIdx;
                    blockSize[wstIdx] -= processSize[i];
                }
            }

            printf("\nProcess No.\tProcess Size\tBlock no.\n");
            for (int i = 0; i < b1; i++)
            {
                printf("%d\t\t%d\t\t",i+1,processSize[i]);
                if (allocation2[i] != -1)
                    printf("%d",allocation2[i] + 1);
                else
                   printf( "Not Allocated");
                printf("\n");
            }
}
```

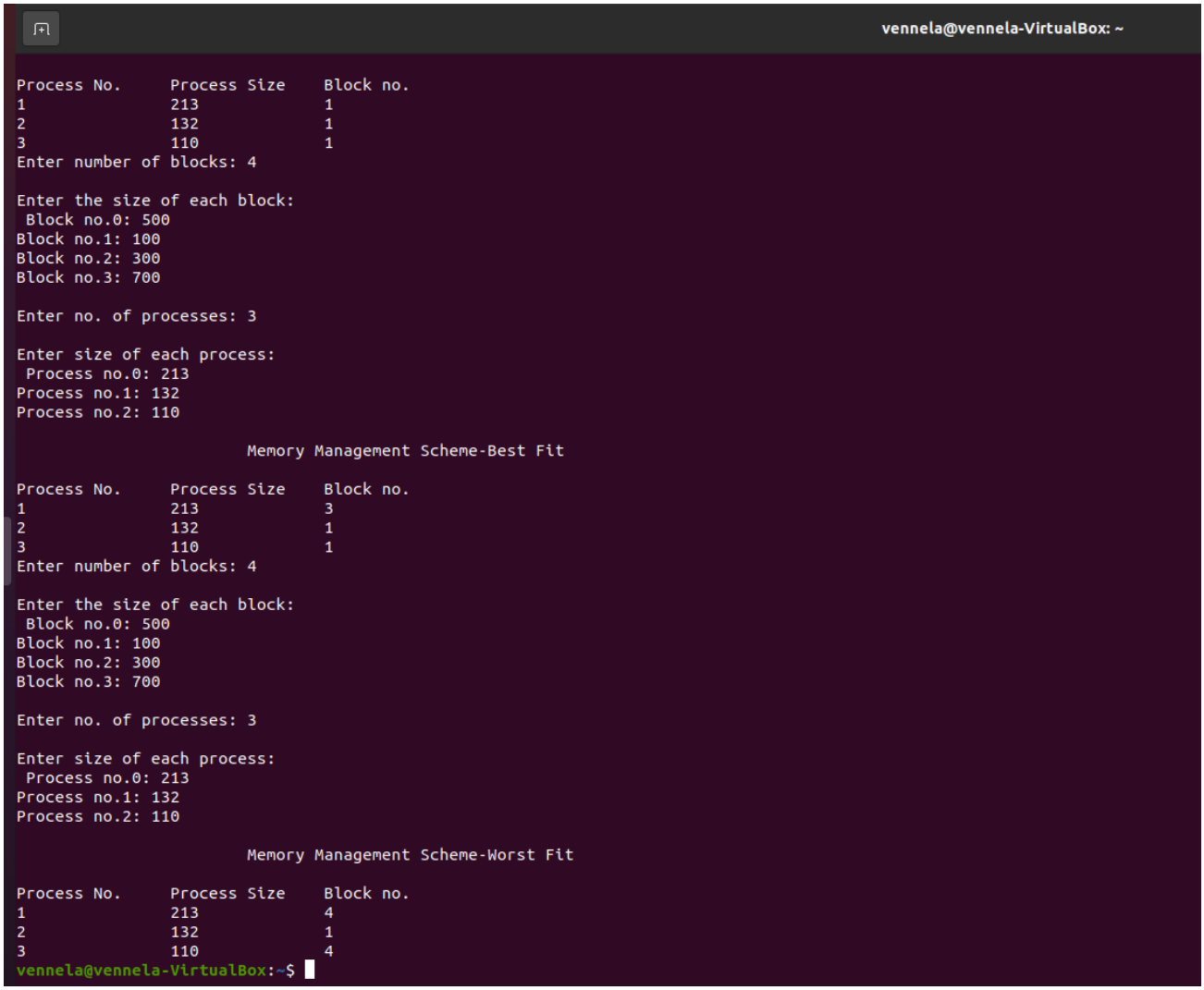
```
int main()
{



firstFit();
 bestFit();
worstFit();
return 0 ;
}
```

OUTPUT:




vennela@vennela-VirtualBox: ~
Enter the size of each block:
Block no.0: 100
Block no.1: 500
Block no.2: 200
Block no.3: 300
Block no.4: 600
Enter no. of processes: 4
Enter size of each process:
Process no.0: 212
Process no.1: 417
Process no.2: 112
Process no.3: 426
Memory Management Scheme-Best Fit
Process No. Process Size Block no.
1 212 4
2 417 2
3 112 3
4 426 5
Enter number of blocks: 5
Enter the size of each block:
Block no.0: 100
Block no.1: 500
Block no.2: 200
Block no.3: 300
Block no.4: 600
Enter no. of processes: 4
Enter size of each process:
Process no.0: 212
Process no.1: 417
Process no.2: 112
Process no.3: 426
Memory Management Scheme-Worst Fit
Process No. Process Size Block no.
1 212 5
2 417 2
3 112 5
4 426 Not Allocated
vennela@vennela-VirtualBox:~$


vennela@vennela-VirtualBox: ~
vennela@vennela-VirtualBox:~$ gcc q5.c
vennela@vennela-VirtualBox:~$ ./a.out
Enter number of blocks: 4
Enter the size of each block:
Block no.0: 500
Block no.1: 100
Block no.2: 300
Block no.3: 700
Enter no. of processes: 3
Enter size of each process:
Process no.0: 213
Process no.1: 132
Process no.2: 110
Memory Management Scheme-First Fit
Process No. Process Size Block no.
1 213 1
2 132 1
3 110 1
Enter number of blocks: 4
Enter the size of each block:
Block no.0: 500
Block no.1: 100
Block no.2: 300
Block no.3: 700
Enter no. of processes: 3
Enter size of each process:
Process no.0: 213
Process no.1: 132
Process no.2: 110
Memory Management Scheme-Best Fit
Process No. Process Size Block no.
1 213 3
2 132 1
3 110 1
Enter number of blocks: 4
Enter the size of each block:
Block no.0: 500


vennela@vennela-VirtualBox: ~
Process No. Process Size Block no.
1 213 1
2 132 1
3 110 1
Enter number of blocks: 4
Enter the size of each block:
Block no.0: 500
Block no.1: 100
Block no.2: 300
Block no.3: 700
Enter no. of processes: 3
Enter size of each process:
Process no.0: 213
Process no.1: 132
Process no.2: 110
Memory Management Scheme-Best Fit
Process No. Process Size Block no.
1 213 3
2 132 1
3 110 1
Enter number of blocks: 4
Enter the size of each block:
Block no.0: 500
Block no.1: 100
Block no.2: 300
Block no.3: 700
Enter no. of processes: 3
Enter size of each process:
Process no.0: 213
Process no.1: 132
Process no.2: 110
Memory Management Scheme-Worst Fit
Process No. Process Size Block no.
1 213 4
2 132 1
3 110 4
vennela@vennela-VirtualBox:~$