**ASSIGNMENT**

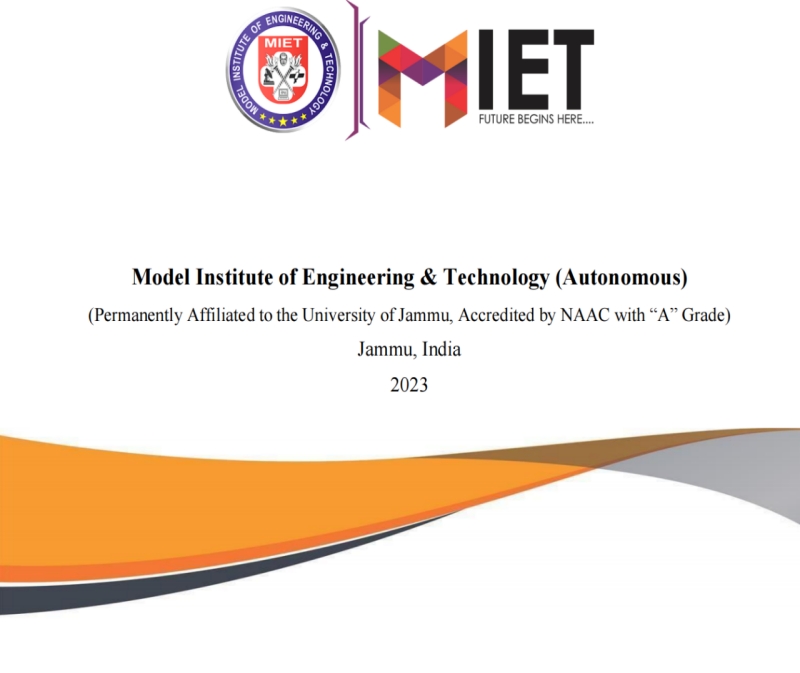
**By**

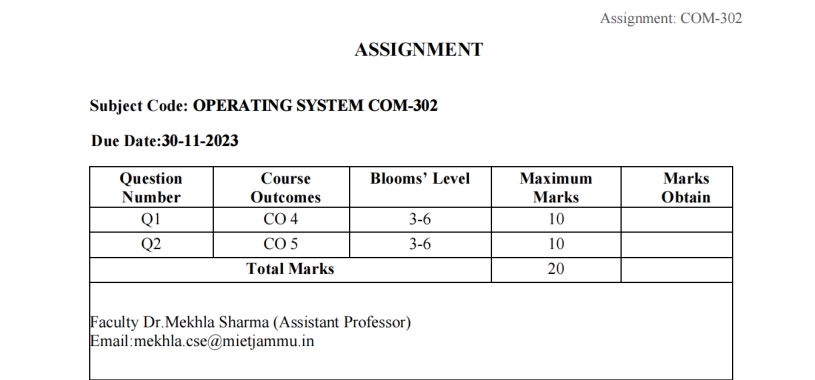
***Raghav Choudhary***

**2022A1r040**

**3rd**

**C.S.E**





**Index**

|  |  |  |
| --- | --- | --- |
| **S.NO** | **TASK** | **P.NO** |
| **1** | Design a program that implements Round Robin scheduling Algorithm. Create a set of processes with specified quantum time and demonstrate how the operating system schedules these processes. Implement and analyze the algorithm with at least 3 different specified quantum time. | **3 - 11** |
| **2** | Design and implement various methods for IPC, such as message passing or shared memory, to facilitate communication between processes in the operating system. | **12-16** |

**Assignment Objectives of**

**Task 1:** Design a program that implements Round Robin scheduling Algorithm. Create a set of processes with specified quantum time and demonstrate how the operating system schedules these processes. Implement and analyze the algorithm with at least 3 different specified quantum time.

*Objective:*

This task main objective is to introduce the Round Robin scheduling algorithm, which is the essential part of an operation system. This goal will let us understand processes scheduling and time-sharing systems. The exercises with round robot algorithm give participants hands-on experience of designing algorithms for an operating system and solving algorithmic problems.

*Learning Outcome:*

We shall will be able to comprehend the idea behind time-slicing and why the algorithm apportions CPU time to processes cyclically. The learning outcome includes a practical sense of coding, implementation, and evaluating Round Robin algorithm with distinct quantum times. Varying the quantum time allows us to understand the effect of response times, cycle times, and processing times on the system.

**Task 2:** Design and implement various methods for IPC, such as message passing or shared memory, to facilitate communication between processes in the operating system.

*Learning Objective:*

In this exercise, participant will learn about designing IPC functions in OS. At this point, the objective is to give participants deeper appreciation on how the techniques used in IPC like message passing and shared memory allow processes to communicate with one another. Participants will practically learn how to overcome process communication challenges in a complex environment through the involvement in design and implementation process.

*Learning Outcome:*

Once the task is completed, participants should be able to attain some learning outcomes. At last, they will be able to develop various IPC techniques; demonstrate to the fact that they understood the concept behind the message switching mechanism and shared memory

**Task 1**

Design a program that implements Round Robin scheduling Algorithm. Create a set of processes with specified quantum time and demonstrate how the operating system schedules these processes. Implement and analyze the algorithm with at least 3 different specified quantum time.

**Solution:** The Round robin scheduling algorithm is one of the CPU scheduling algorithms in which every process gets a fixed amount of time quantum to execute the process. In this algorithm, every process gets executed cyclically. This means that processes that have their burst time remaining after the expiration of the time quantum are sent back to the ready state and wait for their next turn to complete the execution until it terminates. This processing is done in FIFO order which suggests that processes are executed on a first-come, first-serve basis.

***Algorithm for Round-Robin Code***

Step 1: Organize all processes according to their arrival time in the ready queue. The queue structure of the ready queue is based on the FIFO structure to execute all CPU processes.

Step 2: Now, push the first process from the ready queue to execute its task for a fixed time, allocated by each process that arrives in the queue.

Step 3: If the process cannot complete their task within defined time interval or slots because it is stopped by another process that pushes from the ready queue to execute their task due to arrival time of the next process is reached. Therefore, CPU saved the previous state of the process, which helps to resume from the point where it is interrupted. (If the burst time of the process is left, push the process end of the ready queue).

Step 4: Similarly, the scheduler selects another process from the ready queue to execute its tasks. When a process finishes its task within time slots, the process will not go for further execution because the process's burst time is finished.

Step 5: Similarly, we repeat all the steps to execute the process until the work has finished.

a.Design a program that implements Round Robin scheduling Algorithm

#include<stdio.h>

void rr(int at[],int n,int bt[],int qt){

int rt[n];

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

rt[i] = bt[i];

int ct=0;

while(1){

int done = 1;

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

if(rt[i]>0){

done = 0;

if(rt[i]>qt){

ct += qt;

rt[i] -= qt;

printf("The Process ID NUMBER %d has executed for specified time quantum.Remaining Time: %d\n",i+1,rt[i]);

}

else{

ct+= rt[i];

rt[i] = 0;

printf("Process ID NUMBER %d is completed.Execution Time: %d\n",i+1,ct);

}

}

}

if(done == 1)

break;

}

}

int main()

{

int n;

printf("ENTER NUMBER OF PROCESSES : ");

scanf("%d",&n);

int at[n];

int bt[n];

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

printf("ENTER ARRIVAL-TIME OF PROCESS ID %d: ",i+1);

scanf("%d",&at[i]);

printf("ENTER BURST TIME OF PROCESS ID %d : ",i+1);

scanf("%d",&bt[i]);

}

int qt;

printf("ENTER QUANTUM TIME FOR YOUR PROCESSES: ");

scanf("%d",&qt);

rr( at, n,bt, qt);

return 0;

}

b) Create a set of processes with specified quantum time and demonstrate how the operating system schedules these processes. Implement and analyze the algorithm with at least 3 different specified quantum time

|  |  |  |
| --- | --- | --- |
| Process ID | Arrival Time | Burst Time |
| 1 | 0 | 5 |
| 2 | 1 | 6 |
| 3 | 2 | 3 |
| 4 | 3 | 1 |
| 5 | 4 | 5 |
| 6 | 6 | 4 |

Example 1: There are six processes named as P1, P2, P3, P4, P5 and P6. Their arrival time and burst time are given below in the table. The time quantum of the system is 4 units.

Gant Chart after its exectution would be:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| P1 | P2 | P3 | P4 | P5 | P1 | P6 | P2 | P5 |

0 4 8 11 16 17 21 23 24

Steps:

1)The P1 will be executed for 4 units first. Meanwhile the execution of P1, four more processes P2, P3, P4 and P5 arrives in the ready queue. P1 has not completed yet, it needs another 1 unit of time hence it will also be added back to the ready queue

Gantt Chart would be

|  |
| --- |
| P1 |

1. 4

Ready Queue would be

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| P2 | P3 | P4 | P5 | P1 |
| 6 | 3 | 1 | 5 | 1 |

2) After P1, P2 will be executed for 4 units

Gantt Chart

|  |  |
| --- | --- |
| P1 | P2 |

0 4 8

Ready queue During the execution of P2, one more process P6 is arrived in the ready queue. Since P2 has not completed yet hence, P2 will also be added back to the ready queue with the remaining burst time 2 units.

Ready queue would be

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| P3 | P4 | P5 | P1 | P6 | P2 |
| 3 | 1 | 5 | 1 | 4 | 2 |

3) After P1 and P2, P3 will get executed for 3 units of time since its CPU burst time is only 3 seconds.

Gantt Chart

|  |  |  |
| --- | --- | --- |
| P1 | P2 | P3 |

0 4 8 11

Ready Queue : Since P3 has been completed, hence it will be terminated and not be added to the ready queue. The next process will be executed is P4.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| P4 | P5 | P1 | P6 | P2 |
| 1 | 5 | 1 | 4 | 2 |

4) After, P1, P2 and P3, P4 will get executed. Its burst time is only 1 unit which is lesser then the time quantum hence it will be completed.

Gantt Chart

|  |  |  |  |
| --- | --- | --- | --- |
| P1 | P2 | P3 | P4 |

0 4 8 11 12

Ready Queue : The next process in the ready queue is P5 with 5 units of burst time. Since P4 is completed hence it will not be added back to the queue.

|  |  |  |  |
| --- | --- | --- | --- |
| P5 | P1 | P6 | P2 |
| 5 | 1 | 4 | 2 |

5) P5 will be executed for the whole time slice because it requires 5 units of burst time which is higher than the time slice.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| P1 | P2 | P3 | P4 | P5 |

0 4 8 11 12 16

Ready Queue : P5 has not been completed yet; it will be added back to the queue with the remaining burst time of 1 unit

|  |  |  |  |
| --- | --- | --- | --- |
| P1 | P6 | P2 | P5 |
| 1 | 4 | 2 | 1 |

6) The process P1 will be given the next turn to complete its execution. Since it only requires 1 unit of burst time hence it will be completed.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| P1 | P2 | P3 | P4 | P5 | P1 |

0 4 8 11 12 16 17

Ready Queue : P1 is completed and will not be added back to the ready queue. The next process P6 requires only 4 units of burst time and it will be executed next.

|  |  |  |
| --- | --- | --- |
| P6 | P2 | P5 |
| 4 | 2 | 1 |

7) P6 will be executed for 4 units of time till completion.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| P1 | P2 | P3 | P4 | P5 | P1 | P6 |

0 4 8 11 12 16 17 21

Ready Queue: Since P6 is completed, hence it will not be added again to the queue. There are only two processes present in the ready queue. The Next process P2 requires only 2 units of time.

|  |  |
| --- | --- |
| P2 | P5 |
| 2 | 1 |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| P1 | P2 | P3 | P4 | P5 | P1 | P6 | P2 |

8) P2 will get executed again, since it only requires only 2 units of time hence this will be completed.

0 4 8 11 12 16 17 21 23

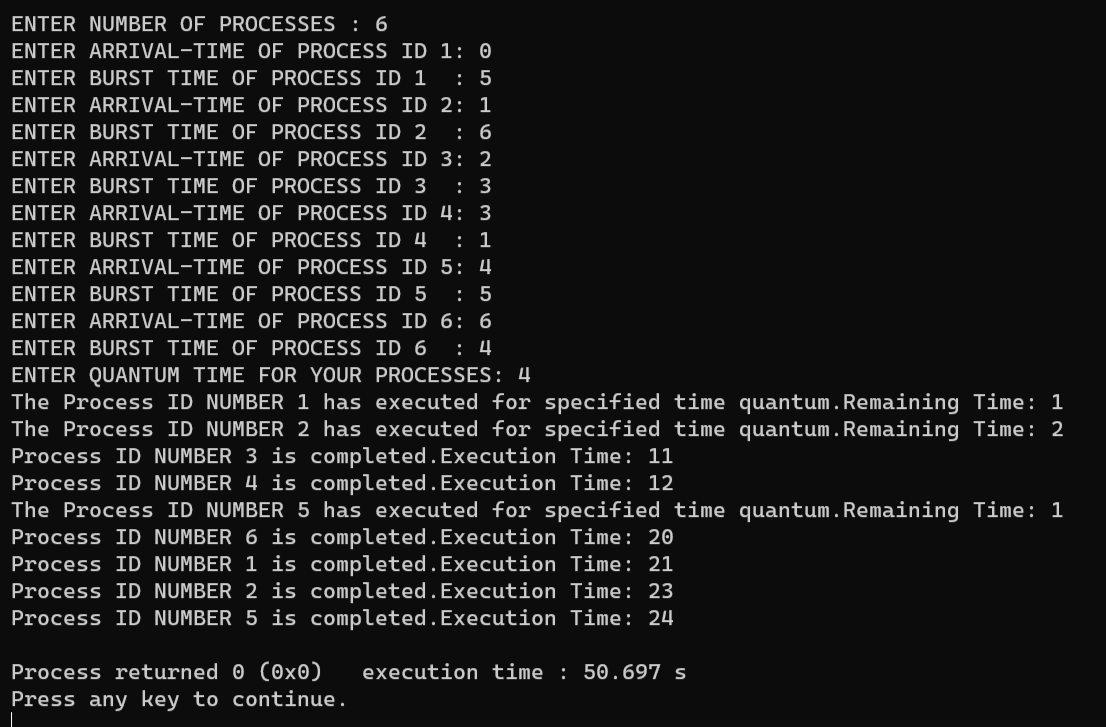
Ready Queue: Only P5 remaining

9) P5 will get executed till completion.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| P1 | P2 | P3 | P4 | P5 | P1 | P6 | P2 | P5 |

0 4 8 11 12 16 17 21 23 24

**Code Output**



Example 2: Consider this following three processes.Their arrival time and burst time are given below in the table. The time quantum of the system is 2 units**.**

|  |  |  |
| --- | --- | --- |
| Process ID | Arrival Time | Burst Time |
| P1 | 0 | 4 |
| P2 | 1 | 3 |
| P3 | 2 | 5 |

Gant Chart after its exectution would be:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| P1 | P2 | P3 | P1 | P2 | P3 | P3 |

0 2 4 6 8 9 11 12

Steps:

1)P1 would be executed first. Remaining time for P1 would be 2 . So Ready queue would be P2,P3,P1.

2) P2 would be executed then .Remaining time for P2 would be 1.So ready queue would be P3,P1,P2.

3) P3 would be executed then .Remaining time for P3 would be 3.So ready queue would be P1,P2,P3.

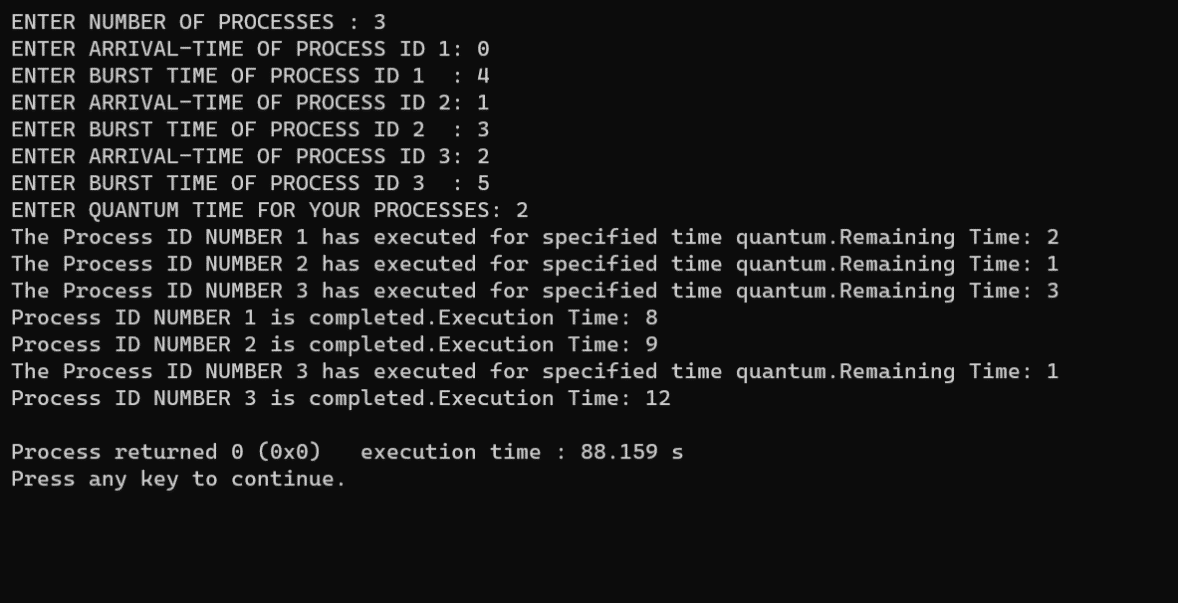
4) P1 would be completely executed then as quantum time is more than the remaining time .So ready queue would be P2,P3.

5) P2 would be completely executed then as quantum time is more than the remaining time .So ready queue would be P3.

6) P3 would be executed then .Remaining time for P3 would be 1.So ready queue would be P3.

7) P2 would be completely executed then as quantum time is more than the remaining time .

**Output of Code**



Example 3 : Consider this following three processes.Their arrival time and burst time are given below in the table. The time quantum of the system is 5 units.

|  |  |
| --- | --- |
| Process Id | Burst Time |
| P1 | 21 |
| P2 | 3 |
| P3 | 6 |
| P4 | 2 |

Solution: Gantt Chart after its execution would be

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| P1 | P2 | P3 | P4 | P1 | P3 | P1 | P1 | P1 |

0 5 8 13 15 20 21 26 31 32

Steps:

1.As arrival time is same for all the processes so the queue would be P1,P2,P3,P4.

2.P1 would be executed first. Remaining time for P1 would be 16.Ready queue would be P2,P3,P4,P1.

3. P2 would be completely executed then as quantum time is more than the remaining time .So ready queue would be P3,P4,P1.

4.P3 would be executed then .Remaining time for P3 would be 1.So ready queue would be P4,P1,P3.

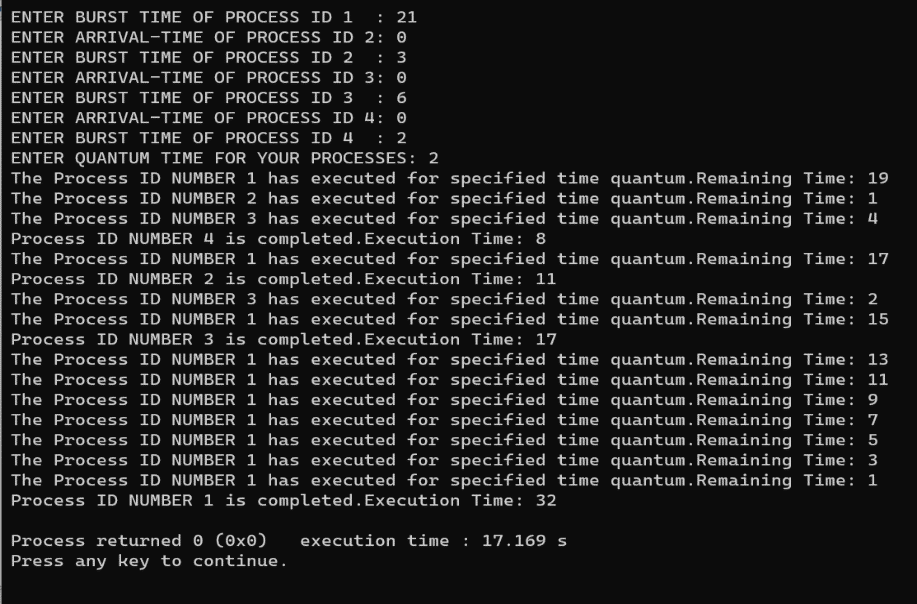
5. P4 would be completely executed then as quantum time is more than the remaining time .So ready queue would be P1,P3.

6. P1 would be executed then .Remaining time for P1 would be 11.So ready queue would be P3,P1.

7. P3 would be completely executed then as quantum time is more than the remaining time .So ready queue would be P1.

8.As only one process is remaining so execute it till it get completely executed.

**Code Screenshot**



**Task 2**

Design and implement various methods for IPC, such as message passing or shared memory, to facilitate communication between processes in the operating system.

Solution: Inter-process communication (IPC) is a mechanism that allows processes to communicate with each other and synchronize their actions. The communication between these processes can be seen as a method of co-operation between them. Processes can communicate with each other through both:

1.Message passing

2.Shared Memory

a.Message Passing : Message passing systems involve communicating processes that share information between them through a channel. Message passing involves two operations: sending and receiving messages. A unit of data which has information relating to communication is known as a message. The message itself may take on various sizes which are either variable or fixed, having some prescribed format. Messages meant for a particular process can be sent using the process identifier while group messages may employ the use of a broadcast or multicast mechanism. The message could be sent to a process that indicates the identifier of the process that it is coming from or use a wildcard or a condition. Synchronous message passing would block the sender’s and the receiver’s subsequent execution after the operation, for instance. Asynchronous message passing, however, does not halt the executions of both devices.

b.Shared Memory : Shared Memory is the fastest inter-process communication (IPC) method. The operating system maps a memory segment in the address space of several processes so that those processes can read and write in that memory segment.

Two functions: shmget() and shmat() are used for IPC using shared memory. shmget() function is used to create the shared memory segment while shmat() function is used to attach the shared segment with the address space of the process.

The first parameter specifies the unique number (called key) identifying the shared segment. The second parameter is the size of the shared segment e.g. 1024 bytes or 2048 bytes. The third parameter specifies the permissions on the shared segment. On success the shmget() function returns a valid identifier while on failure it return -1.

Syntax (shmat()):

#include <sys/types.h>

#include <sys/shm.h>

void \*shmat(int shmid, const void \*shmaddr, int shmflg);

shmat() is used to attach the created shared segment with the address space of the calling process. The first parameter is the identifier which shmget() function returns on success. The second parameter is the address where to attach it to the calling process. A NULL value of second parameter means that the system will automatically choose a suitable address. The third parameter is ‘0’ if the second parameter is NULL, otherwise, the value is specified by SHM\_RND.

Two program for IPC using shared memory. Program 1 will create the shared segment, attach to it and then write some content into it. Then Program 2 will attach itself to the shared segment and read the value written by Program 1.

***Program 1:*** This program creates a shared memory segment, attaches itself to it and then writes some content into the shared memory segment.

#include<stdio.h>

#include<stdlib.h>

#include<unistd.h>

#include<sys/shm.h>

#include<string.h>

int main()

{

int i;

void \*shared\_memory;

char buff[100];

int shmid;

shmid=shmget((key\_t)2345, 1024, 0666|IPC\_CREAT);

printf("Key of shared memory is %d\n",shmid);

shared\_memory=shmat(shmid,NULL,0);

printf("Process attached at %p\n",shared\_memory);

printf("Enter some data to write to shared memory\n");

read(0,buff,100); //get some input from user

strcpy(shared\_memory,buff); //data written to shared memory

printf("You wrote : %s\n",(char \*)shared\_memory);

}

Working of Program 1

shmget() function creates a segment with key 2345, size 1024 bytes and read and write permissions for all users. It returns the identifier of the segment which gets store in shmid. This identifier is used in shmat() to attach the shared segment to the address space of the process. NULL in shmat() means that the OS will itself attach the shared segment at a suitable address of this process.Then some data is read from the user using read() system call and it is finally written to the shared segment using strcpy() function.

Explaination of lines of code :

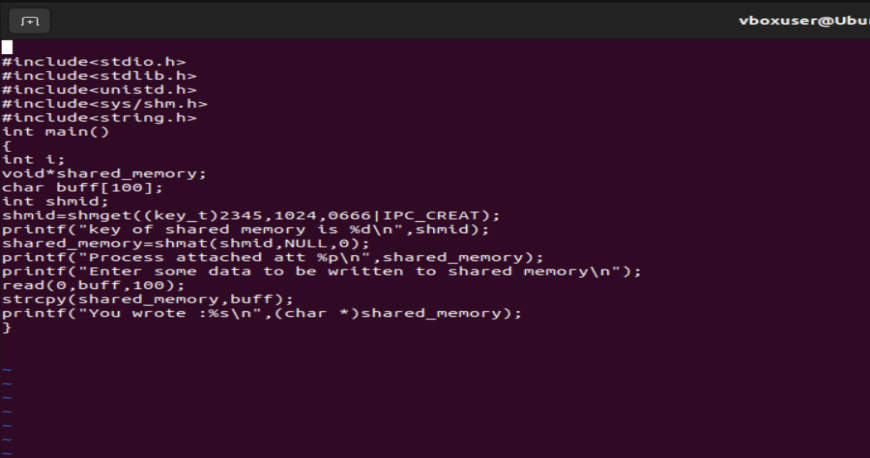
shmid=shmget((key\_t)2345, 1024, 0666|IPC\_CREAT); -creates shared memory segment with key 2345, having size 1024 bytes. IPC\_CREAT is used to create the shared segment if it does not exist. 0666 are the permisions on the shared

segmentshared\_memory=shmat(shmid,NULL,0); //process attached to shared memory

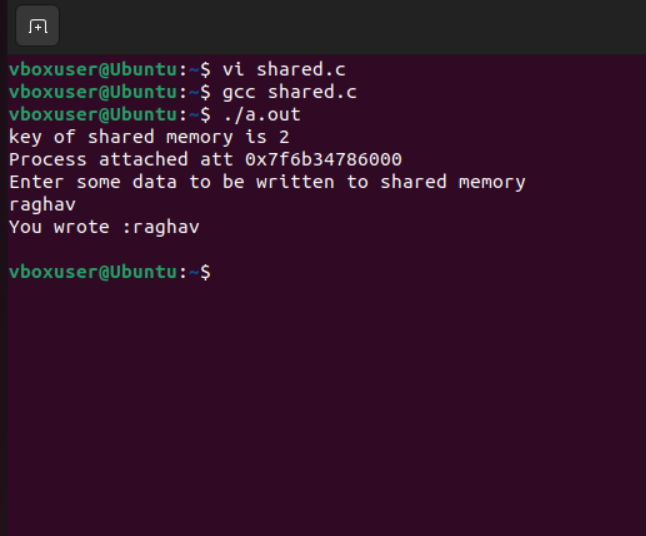
segmentprintf("Process attached at %p\n",shared\_memory); -this prints the address where the segment is attached with this process

printf("Enter some data to write to shared memory\n");&read(0,buff,100);-get some input from user

**Code Screenshot**



**Output**



**Program 2:** This program attaches itself to the shared memory segment created in Program 1. It reads the content of the shared memory

#include<stdio.h>

#include<stdlib.h>

#include<unistd.h>

#include<sys/shm.h>

#include<string.h>

int main()

{

int i;

void \*shared\_memory;

char buff[100];

int shmid;

shmid=shmget((key\_t)2345, 1024, 0666);

printf("Key of shared memory is %d\n",shmid);

shared\_memory=shmat(shmid,NULL,0); //process attached to shared memory segment

printf("Process attached at %p\n",shared\_memory);

printf("Data read from shared memory is : %s\n",(char \*)shared\_memory);

}

Code :



Output:

