Operating Systems Lab
Practical Examination
Assignment 15
(from chit code)

Name: Aditya Kangune
Roll number: 33323
Batch: K11
Date: 17/12/2021

Aim:

Inter-process Communication using Shared Memory using System V. Application to demonstrate: Client and Server Programs in which server process creates a shared memory segment and writes the message to the shared memory segment. The client process reads the message from the shared memory segment and displays it to the screen

Programs:

server.c

```
// COMPILE : gcc server.c -o server

// EXECUTE : ./server



#include<stdio.h>

#include<unistd.h>

#include<stdlib.h>

#include<sys/shm.h>

#include<string.h>
```

```c
int main() {

    int i;

    void *shared_memory;

    char buff[100];

    int shmid;



    // shmget - Used to create shared memory segment.

    // shmget(key_t key, size_t size, int shmflg)

        // key_t - unique number (key) for identifying the
shared segment

        // size_t - size of the shared segment e.g. 1024 or
2048 bytes

        // shmflg - permissions on the shared segment

    // On success, shmget returns a valid identifier or
else -1 on failure.

    shmid = shmget((key_t)2345, 1024, 0666 | IPC_CREAT); //
Creates shared memory with key 20345, having size 1024
bytes.

    // IPC_CREAT is used to create the shared segment if it
does not exist



    printf("\n ++++++++++++++++++++++++++++++++++
Inter-process Communication using Shared Memory
++++++++++++++++++++++++++++++++++ \n\n");

    printf("\n\t ------------------------------------
SERVER ------------------------------------ \n\n");
```

```
    printf("\t\tKey of shared memory is: %d\n\n", shmid);


    // shmat - Used to attach the shared segment with the
address space of the calling process.
    shared_memory=shmat(shmid, NULL, 0);
    // void *shmat(int, shmid, const void *shmaddr, int
shmflg);
        // shmid - identifier that shmget() returns on
success.
        // shmaddr - address where to attach it to the
calliing process
            // if shmaddr is NULL, it means system will
automatically chose a suitable ddress.
        // smgflg - is 0 is second parameter is NULL,
otherwise thevalue is specified by SHM_RND


    printf("\t\tProcess attached at %p\n\n",
shared_memory);


    // ssize_t read(int fd, void *buf, size_t count);
    // read() attempts to read up to count bytes from file
descriptor fd into the buffer starting at buf.


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


    strcpy(shared_memory, buff); // data written to shared
memmory
```

```
    printf("\t\tYou wrote: %s\n\n", (char*) shared_memory);

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



    printf("\n \t------------------------------------
SERVER ENDS ------------------------------------ \n\n");



}
```

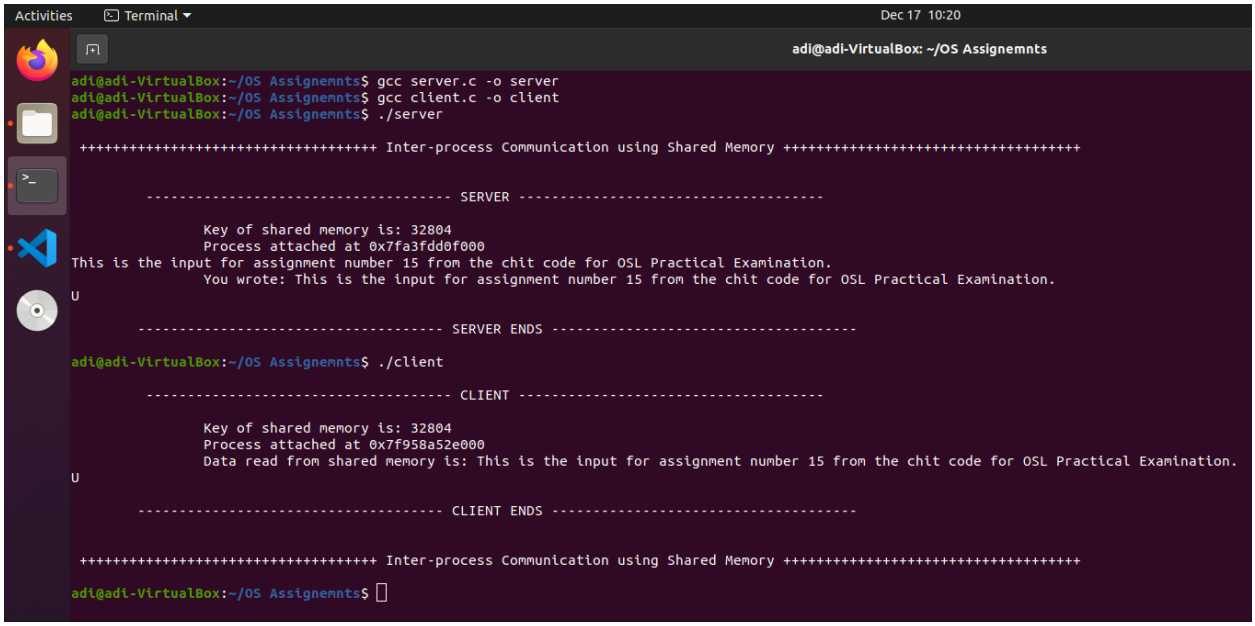
client.c

```
// COMPILE : gcc client.c -o client

// EXECUTE : ./client



#include<stdio.h>

#include<unistd.h>

#include<stdlib.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("\n\t ------------------------------------
CLIENT ------------------------------------ \n\n");

    printf("\t\tKey of shared memory is: %d\n\n", shmid);

    shared_memory=shmat(shmid, NULL, 0); // process
attached to shared memorysegment



    printf("\t\tProcess attached at %p\n\n",
shared_memory);

    printf("\t\tData read from shared memory is: %s\n",
(char*) shared_memory);



    printf("\n \t-------------------------------------
CLIENT ENDS ------------------------------------- \n\n");




    printf("\n +++++++++++++++++++++++++++++++++++
Inter-process Communication using Shared Memory
+++++++++++++++++++++++++++++++++++ \n\n");

}
```

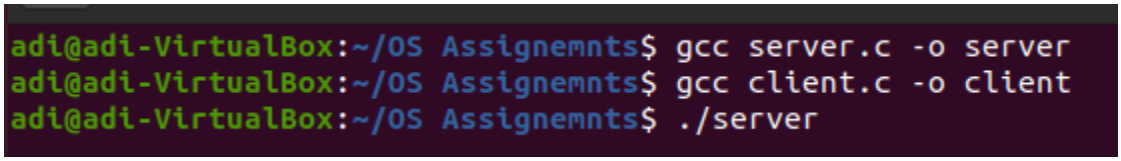
Output:

Entire Screenshot:

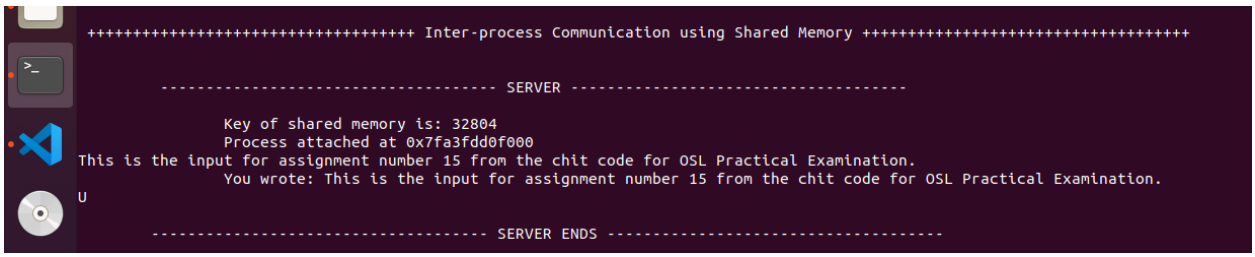


Stepwise execution:

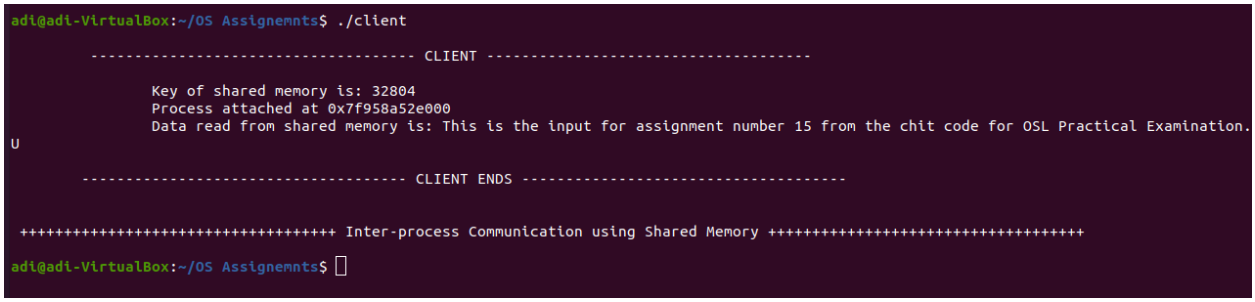
1) Compiling server.c and client.c
2) Executing server.c:



3) Entering input in the server program:



4) Executing client.c:



Conclusion:

- Assignment 15 was successfully executed.
- The client and server processes were able to share the same buffer memory.
- The **unique key** of memory for server and client is seen to be the **same** as they have shared the same memory.
- Whereas the **process address** for client and server is seen to be **different** as both are different processes.
- Hence two different processes were able to share the same **memory of size 1024 bytes.**

---------- X ----------