TOPIC: Implementation of Shared Memory and its system calls.

THEORY:

In computer science, inter-process communication or inter-process communication (IPC) refers specifically to the mechanisms an operating system provides to allow processes it manages to share data. Typically, applications can use IPC categorized as clients and servers, where the client requests data and the server responds to client requests.

SHARED MEMORY:

Shared memory is a segment of memory that is shared between processes. It is an efficient means of passing data between programs. One program will create a memory portion which other processes (if permitted) can access.

Creating the segment and connecting:

A shared memory segment is created and connected to via the shmget() call:

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

Upon successful completion, shmget() returns an identifier for the shared memory segment. The key argument should be created the same was as shown in the Message Queues document, using ftok(). The next argument, size, is the size in bytes of the shared memory segment. Finally, the shmflg should be set to the permissions of the segment bitwise-ORd with IPC_CREAT if you want to create the segment, but can be 0 otherwise.

Attach me—getting a pointer to the segment

Before you can use a shared memory segment, you have to attach yourself to it using the shmat() call:

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

shmid is the shared memory ID you got from the call to shmget(). Next is shmaddr, which you can use to tell shmat() which specific address to use but you should just set it to 0 and let the OS choose the address for you. Finally, the shmflg can be set to SHM_RDONLY if you only want to read from it, 0 otherwise.

Detaching from the segments

When you're done with the shared memory segment, your program should detach itself from it using the shmdt() call:

int shmdt(void *shmaddr);

The only argument, shmaddr, is the address you got from shmat(). The function returns -1 on error, 0 on success.

Deleting the segments

When you detach from the segment, it isn't destroyed. Nor is it removed when everyone detaches from it. You have to specifically destroy it using a call to shmctl(), similar to the control calls for the other System V IPC functions:

shmctl(shmid, IPC_RMID, NULL);

The above call deletes the shared memory segment, assuming no one else is attached to it.

Advantages of Shared memory

- fastest
- flexible
- doesn't require continuous kernel interruption.

IMPLEMENTATION:

SERVER-SIDE CODE:

```
// shm_server.c
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <stdio.h>
#include<stdlib.h>
#define SHMSZ 27
main()
{
        char c;
        int shmid;
        key_t key;
        char *shm, *s;
        /*
        * We'll name our shared memory segment
        * "5678".
        */
        key = 5678;
        /*
        * Create the segment.
        */
        if ((shmid = shmget(key, SHMSZ, IPC_CREAT | 0666)) < 0) {
        perror("shmget");
        exit(1);
        }
        /*
        * Now we attach the segment to our data space.
        */
        if ((shm = shmat(shmid, NULL, 0)) == (char *) -1) {
        perror("shmat");
        exit(1);
        }
        /*
        * Now put some things into the memory for the
        * other process to read.
```

```
        */
        s = shm;
        for (c = 'a'; c <= 'z'; c++)
        *s++ = c;
        *s = 0;
        /*
        * Finally, we wait until the other process
        * changes the first character of our memory
        * to '*', indicating that it has read what
        * we put there.
        */
        while (*shm != '*')
        sleep(1);
        exit(0);
}
```

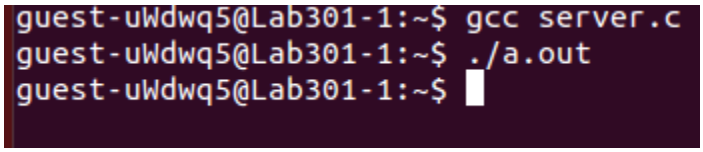
CLIENT-SIDE CODE:

```
// shm_client.c
/*
 * shm-client - client program to demonstrate shared memory.
 */
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <stdio.h>
#include<stdlib.h>
#define SHMSZ  27
main()
{
        int shmid;
        key_t key;
        char *shm, *s;
        /*
        * We need to get the segment named
        * "5678", created by the server.
        */
        key = 5678;
        /*
        * Locate the segment.
        */
        if ((shmid = shmget(key, SHMSZ, 0666)) < 0) {
        perror("shmget");
```

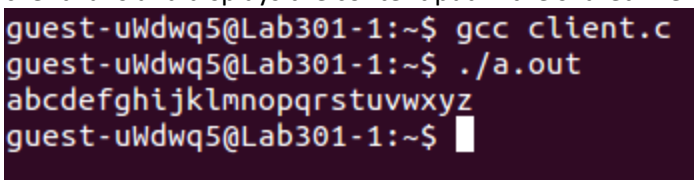
```
        exit(1);
        }
        /*
        * Now we attach the segment to our data space.
        */
        if ((shm = shmat(shmid, NULL, 0)) == (char *) -1) {
        perror("shmat");
        exit(1);
        }
        /*
        * Now read what the server put in the memory.
        */
        for (s = shm; *s != 0; s++)
        putchar(*s);
        putchar('\n');
        /*
        * Finally, change the first character of the
        * segment to '*', indicating we have read
        * the segment.
        */
        *shm = '*';
        exit(0);
}
```

OUTPUT:

First,the server runs and becomes blocked until client completes its execution.



client runs and displays the content put in the shared memory by the server.



CONCLUSION:

The shared memory system calls like shmget,shmat etc are used while using shared memory as an IPC model.
It should be ensured that same 'key' value is given to both client and server.
It is a good practice to destroy the memory segment using shmctl after use to free up memory.
Thus, client server model was implemented using shared memory.