EXPERIMENT NO:6
IPC USING SHARED MEMORY

Aim : Demonstrate inter-process communication via shared memory, with one process writing data to the shared memory segment (sender) and another process reading the data from the same segment (receiver).

Algorithm -writer or sender

Step 1: Start

Step 2: Include necessary header files

Step 3: Define the main() function.

Step 4: Declare variables: i , shared_memory as a void pointer to hold the address of the shared memory segment, buff as a character array to store user input. shmid as an integer to hold the shared memory segment ID.

Step 5: Create a shared memory segment using shmget() function with parameters: key_t type casted key value (2345 in this case), Size of the shared memory segment (1024 bytes), Permissions for the shared memory segment (0666|IPC_CREAT). Store the returned shared memory ID in the shmid variable.

Step 6: Print the shared memory key ,and the shmid variable.

Step 7: Attach the process to the shared memory segment:using shmat() function with parameters:shmid, Shared memory address (NULL for letting the system choose). Flags (0 in this case).

Step 8: Print the address where the process is attached to the shared memory.

Step 9: Display a message asking the user to input some data.

9.1 Read input from the user using read system call

Step 10: Copy user input to shared memory:

Use strcpy() function to copy the content of buff to shared_memory.

Step 11: Print the data written to shared memory:

Print the content of shared_memory after casting it to a character pointer.

End the main() function.

Step 10: Stop .

```
//writer or sender
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<sys/shm.h>
#include<string.h>
#include<sys/ipc.h>
int main()
{
int i;
        void *shared_memory;
        char buff[100];
        int shmid;

        //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
        permissions on the shared segment

        shmid=shmget((key_t)2345, 1024, 0666|IPC_CREAT);
        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); //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
```

```
        strcpy(shared_memory,buff); //data written to shared memory
        printf("You wrote : %s\n",(char *)shared_memory);
        }
//Reader or receiver
```

Algorithm

Step 1: Start

Step 2: Define the `main` function.

Step 3: Declare variables:`i` as an integer,`shared_memory` as a void pointer, `buff` as a character array ,`shmid` as an integer.

Step 4: Shared memory operations:

Step 4.1: Obtain the shared memory segment: - Call `shmget()` with key `2345`. - Allocate 1024 bytes of shared memory. - Set permissions to `0666`.

Step 4.2: Print the obtained shared memory key.

Step 5: Attach the current process to the shared memory segment:Call `shmat()` with the shared memory ID (`shmid`) and `NULL` and Set the flag to `0`.

Step 5.1: Print the address at which the process is attached to the shared memory.

Step 6: Read data from shared memory:Cast `shared_memory` to `(char *)` and Print the content.

Step 7: Stop.

```
        #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);
}
```

OUTPUT:

```
ubuntu@l5sys7:~/os$ gcc sharedmemsend.c
ubuntu@l5sys7:~/os$ ./a.out
Key of shared memory is 2
Process attached at 0x7f0b6749c000
Enter some data to write to shared memory
hello World
You wrote : hello World

ubuntu@l5sys7:~/os$ gcc sharedmemreceiver.c
ubuntu@l5sys7:~/os$ ./a.out
Key of shared memory is 2
Process attached at 0x7f8b3885a000
Data read from shared memory is : hello World
```

Explanation to understand:

Inter Process Communication through shared memory is a concept where two or more processes can access the common memory and communication is done via this shared memory where changes made by one process can be viewed by another process.

The problem with pipes, fifo and message queue – is that for two processes to exchange information. The information has to go through the kernel.

- Server reads from the input file.
- The server writes this data in a message using either a pipe, fifo or message queue. ● The client reads the data from the IPC channel,again requiring the data to be copied from the kernel's IPC buffer to the client's buffer.
- Finally, the data is copied from the client's buffer.

Function	Signature	Description
ftok()	key_t ftok()	It is used to generate a unique key.
shmget()	int shmget(key_t key,size_t size, int shmflg);	Upon successful completion, shmget() returns an identifier for the shared memory segment.
shmat()	void *shmat(int shmid ,void *shmaddr ,int shmflg);	Before you can use a shared memory segment, you have to attach yourself to it using shmat(). Here, **shmid** is a shared memory ID and **shmaddr** specifies the specific address to use but we should set it to zero and OS will automatically choose the address.
shmdt()	int shmdt(void *shmaddr);	When you're done with the shared memory segment, your program should detach itself from it using shmdt().
shmctl()	shmctl(int shmid,IPC_RMID,NULL);	When you detach from shared memory, it is not destroyed. So, to destroy shmctl() is used.