OPERATING SYSTEM PROJECT

Implementation Of IPC Using Shared Memory

Name:	***Roll No:***
• ***Avanpreet Singh***	***2021A1R123***
• ***Aman Gupta***	***2021A1R135***
• ***Satyam Luthra***	***2021A1R142***
• ***Mansi Devi***	***2021A1R144***
• ***Payal Totara***	***2021A1R145***

Section: A3

Semester: 3rd

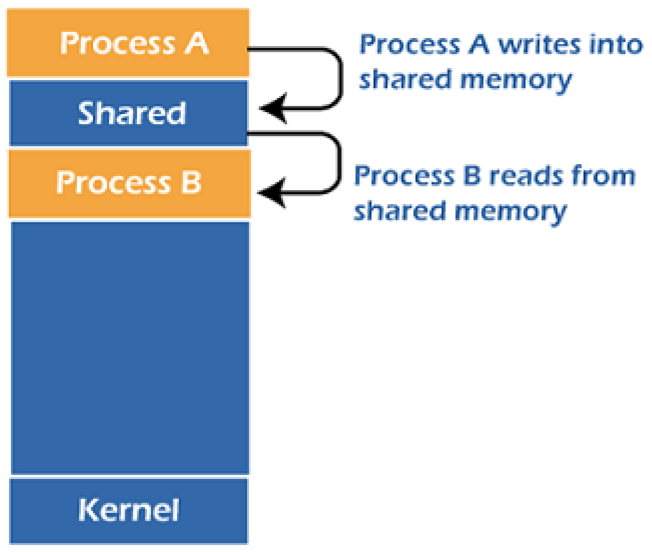
Branch: CSE

Submitted To Ms. Pragti Jamwal

Implementation Of IPC Using Shared Memory

Shared memory is a memory shared between two or more processes. Each process has its own address space; if any process wants to communicate with some information from its own address space to other processes, then it is only possible with IPC (inter-process communication) techniques.

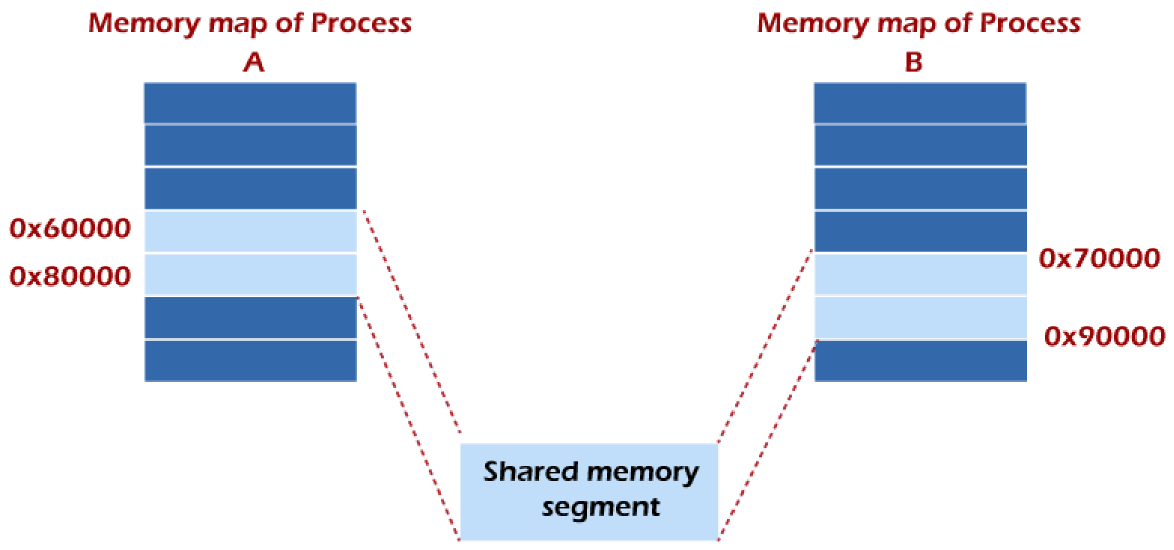
Shared memory is the fastest inter-process communication mechanism. The operating system maps a memory segment in the address space of several processes to read and write in that memory segment without calling operating system functions.


Process A
Shared
Process B
Kernel
Process A writes into shared memory
Process B reads from shared memory

For applications that exchange large amounts of data, shared memory is far superior to message passing techniques like message queues, which require system calls for every data exchange. To use shared memory, we have to perform two basic steps:

- ➢ **Request a memory segment that can be shared between processes to the operating system.**
- ➢ **Associate a part of that memory or the whole memory with the address space of the calling process.**

A shared memory segment is a portion of physical memory that is shared by multiple processes. In this region, processes can set up structures, and others may read/write on them. When a shared memory region is established in two or more processes, there is no guarantee that the regions will be placed at the same base address. Semaphores can be used when synchronization is required.



For example, one process might have the shared region starting at address 0x60000 while the other process uses 0x70000. It is critical to understand that these two addresses refer to the exact same piece of data. So storing the number 1 in the first process's address 0x60000 means the second process has the value of 1 at 0x70000. The two different addresses refer to the exact same location.

Why Choose IPC Through Shared Memory

Usually, inter-related process communication is performed using Pipes or Named Pipes. And unrelated processes communication can be performed using Named Pipes or through popular IPC techniques of Shared Memory and Message Queues.

But the problem with pipes, FIFO, and message queue is that the information exchange between two processes goes through the kernel, and it works as follows.

- ➢ The server reads from the input file.
- ➢ The server writes this data in a message using 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.

A total of four copies of data are required (2 read and 2 write). So, shared memory provides a way by letting two or more processes share a memory segment. With Shared Memory, the data is only copied twice, from the input file into shared memory and from shared memory to the output file.

Functions of IPC Using Shared Memory

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

1. ***shmget() Function***

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 returns -1.

Syntax

- ➢ **#include <sys/ipc.h>**
- ➢ **#include <sys/shm.h>**
- ➢ **int shmget (key_t key, size_t size, int shmflg);**

2. ***shmat() Function***

shmat() function is used to attach the created shared memory segment associated with the shared memory identifier specified by shmid to the calling process's address space. The first parameter here is the identifier which the shmget() function returns on success. The second parameter is the address where to attach it to the calling process. A NULL value of the 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.

Syntax

- ➢ **#include <sys/types.h>**
- ➢ **#include <sys/shm.h>**
- ➢ **void *shmat(int shmid, const void *shmaddr, int shmflg);**

How Does IPC Using Shared Memory Work?

A process creates a shared memory segment using shmget(). The original owner of a shared memory segment can assign ownership to another user with shmctl(). It can also revoke this assignment. Other processes with proper permission can perform various control functions on the shared memory segment using shmctl().

Once created, a shared segment can be attached to a process address space using shmat(). It can be detached using shmdt(). The attaching process must have the appropriate permissions for shmat(). Once attached, the process can read or write to the segment, as the permission requested in the attach operation allows. A shared segment can be attached multiple times by the same process.

A shared memory segment is described by a control structure with a unique ID that points to an area of physical memory. The identifier of the segment is called the shmid. The structure definition for the shared memory segment control structures and prototypes can be found in <sys/shm.h>.

Examples

We will write two programs for IPC using shared memory as an example. Program 1 will create the shared segment, attach it, and then write some content in 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);
- //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
- 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);
- }

How Does It Work?

In the above program, the shmget() function creates a segment with key 2345, size 1024 bytes, and reads and writes permissions for all users. It returns the identifier of the segment, which gets stored in shmid. This identifier is used in shmat() to attach the shared segment to the process's address space.

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 the read() system call, and it is finally written to the shared segment using the strcpy() function.

Program 2: **This program attaches itself to the shared memory segment created in Program 1, and 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);
➢ }
```

How Does It Work?

In this program, shmget() here generates the identifier of the same segment as created in Program 1. Remember to give the same key value. The only change is, do not write IPC_CREAT as the shared memory segment is already created. Next, shmat() attaches the shared segment to the current process.

After that, the data is printed from the shared segment. In the output, you will see the same data that you have written while executing Program 1.