# Ex. 2A IPC Using Shared Memory System Call

**Dr S.Rajarajan APIII/CSE**

## Objective

Develop a program to make a sender and receiver processes communicate using shared memory.

## Introduction

Two running Processes can communicate with each other using these two ways:

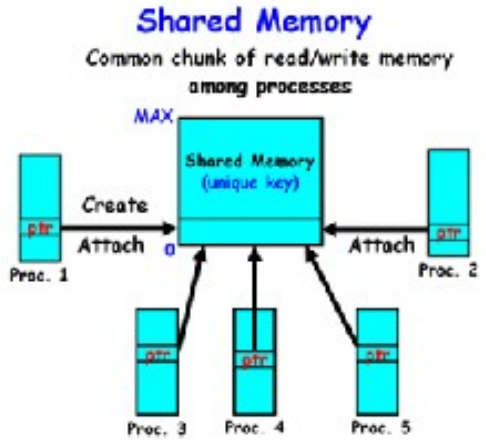
1. Shared Memory
2. Message passing

## Shared Memory

- One of the simplest inter-process communication methods is using shared memory
- Shared memory allows two or more processes to access the same memory as if they all called malloc and were returned pointers to the same actual memory. When one process changes the memory, all the other processes see the modification.

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().** Once created, a shared segment can be attached to a process address space using **shmat().** It can be detached using **shmdt()** . Once attached, the process can read or write to the segment, as allowed by the permission requested in the attach operation.. 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 prototypews can be found in <sys/shm.h>.



**ftok()**: is use to generate a unique key.

**shmget()**: int shmget(key_t,size_tsize,intshmflg); upon successful completion, shmget() returns an identifier for the shared memory segment.

**shmat()**: Before you can use a shared memory segment, you have to attach yourself to it using shmat(). void *shmat(int shmid ,void *shmaddr ,int shmflg); shmid is shared memory id. shmaddr specifies specific address to use but we should set it to zero and OS will automatically choose the address.

**shmdt()**: When you're done with the shared memory segment, your program should detach itself from it using shmdt(). int shmdt(void *shmaddr);
**shmctl()**: when you detach from shared memory,it is not destroyed. So, to destroy shmctl() is used. shmctl(int shmid,IPC_RMID,NULL);

## Allocation

A process allocates a shared memory segment using shmget ("SHared Memory GET"). Its first parameter is an integer key that specifies which segment to create.
Unrelated processes can access the same shared segment by specifying the same key value.

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

The key argument is an access value (key) associated with the shared memory. The size argument is the size in bytes of the requested shared memory. The shmflg argument specifies the initial access permissions and creation control flags.

When the call succeeds, it returns the shared memory segment ID. This call is also used to get the ID of an existing shared segment (from a process requesting sharing of some existing memory portion).

Unfortunately, other processes may have also chosen the same fixed key, which
could lead to conflict. Using the special constant IPC_PRIVATE as the key value guarantees that a brand new memory segment is created.

Its second parameter specifies the number of bytes in the segment. Because segments are allocated using pages, the number of actually allocated bytes is rounded up to an integral multiple of the page size.

The third parameter is the bitwise or of flag values that specify options to shmget.
The flag values include these:

**IPC_CREAT**—This flag indicates that a new segment should be created. This permits creating a new segment while specifying a key value.

**IPC_EXCL**—This flag, which is always used with IPC_CREAT, causes shmget to fail if a segment key is specified that already exists. Therefore, it arranges for the calling process to

have an “exclusive” segment. If this flag is not given and the key of an existing segment is used, shmget returns the existing segment instead of
creating a new one.

**Mode flags**—This value is made of 9 bits indicating permissions granted to owner, group, and world to control access to the segment. Execution bits are ignored. An easy way to specify permissions is to use the constants defined in <sys/stat.h> and documented in the section 2 stat man page.1 For example, S_IRUSR and S_IWUSR specify read and write permissions for the owner of the shared memory segment, and S_IROTH and S_IWOTH specify read and write permissions for others.

For example, this invocation of shmget creates a new shared memory segment (or
access to an existing one, if shm_key is already used) that’s readable and writeable to the owner but not other users.

**int segment_id = shmget (shm_key, getpagesize (), IPC_CREAT | S_IRUSR | S_IWUSER);**

If the call succeeds, shmget returns a segment identifier. If the shared memory segment already exists, the access permissions are verified and a check is made to ensure that the segment is not marked for destruction.

## Controlling a Shared Memory Segment

The shmctl (“SHared Memory ConTroL”) call returns information about a shared
memory segment and can modify it. The first parameter is a shared memory segment identifier.

To obtain information about a shared memory segment, pass **IPC_STAT** as the
second argument and a pointer to a struct shmid_ds.

To remove a segment, pass **IPC_RMID** as the second argument, and pass NULL as the third argument. The segment is removed when the last process that has attached it finally detaches it.

Each shared memory segment should be explicitly deallocated using shmctl when
you’re finished with it, to avoid violating the system wide limit on the total number of shared memory segments. Invoking exit and exec detaches memory segments but does not deallocate them.

shmctl() is used to alter the permissions and other characteristics of a shared memory segment. It is prototyped as follows:

int shmctl(int shmid, int cmd, struct shmid_ds *buf);

The process must have an effective shmid of owner, creator or super user to perform this command. The cmd argument is one of following control commands:

**SHM_LOCK**

-- Lock the specified shared memory segment in memory. The process must have the effective ID of superuser to perform this command.

**SHM_UNLOCK**

-- Unlock the shared memory segment. The process must have the effective ID of superuser to perform this command.

**IPC_STAT**

-- Return the status information contained in the control structure and place it in the buffer pointed to by buf. The process must have read permission on the segment to perform this command.

**IPC_SET**

-- Set the effective user and group identification and access permissions. The process must have an effective ID of owner, creator or superuser to perform this command.

**IPC_RMID**

-- Remove the shared memory segment.

## Attaching and Detaching a Shared Memory Segment

shmat() and shmdt() are used to attach and detach shared memory segments. They are prototypes as follows:

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

int shmdt(const void *shmaddr);

shmat() returns a pointer, shmaddr, to the head of the shared segment associated with a valid shmid. shmdt()detaches the shared memory segment located at the address indicated by shmaddr

## Example two processes comunicating via shared memory: shm_server.c, shm_client.c

We develop two programs here that illustrate the passing of a simple piece of memery (a string) between the processes if running simulatenously:

**shm_server.c**

-- simply creates the string and shared memory portion.

**shm_client.c**

-- attaches itself to the created shared memory portion and uses the string.

***shm_server.c***

```
#include <iostream>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <stdio.h>
using namespace std;
int main()
{
        // ftok to generate unique key
        key_t key = ftok("shmfile",65);
        // shmget creates shared memory and returns an identifier in shmid
        int shmid = shmget(key,1024,0666|IPC_CREAT);
        // shmat to attach to shared memory
        char *str = (char*) shmat(shmid,(void*)0,0);
        cout<<"Write Data : ";
        gets(str);
        printf("Data written in memory: %s\n",str);
        //detach from shared memory
        shmdt(str);
        return 0;
}
```

***shm_client.c***

```
#include <iostream>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <stdio.h>
using namespace std;
int main()
{
        // ftok to generate same key of the server
        key_t key = ftok("shmfile",65);
        // shmget returns an identifier in shmid
        int shmid = shmget(key,1024,0666|IPC_CREAT);
        // shmat to attach to shared memory
        char *str = (char*) shmat(shmid,(void*)0,0);
        printf("Data read from memory: %s\n",str);
        //detach from shared memory
         shmdt(str);
        // destroy the shared memory
        shmctl(shmid,IPC_RMID,NULL);
        return 0;

}
```