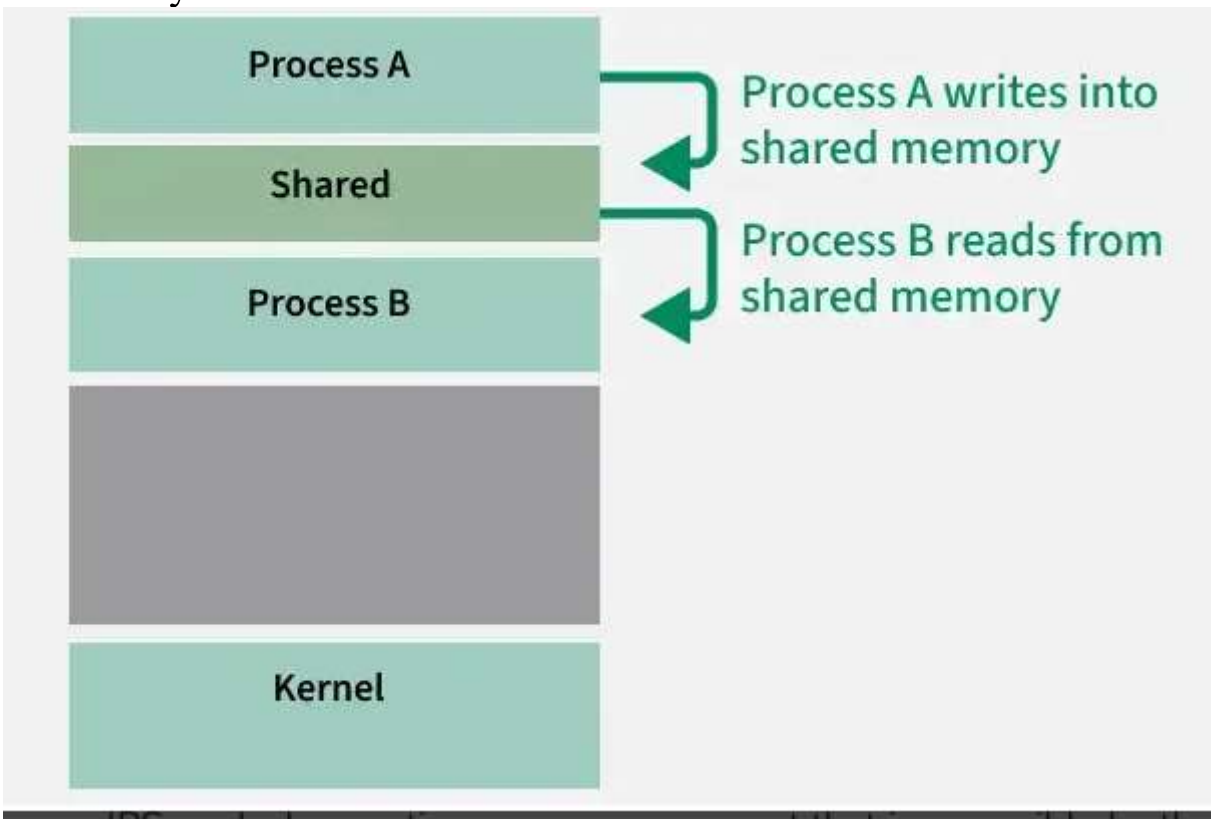
IPC through shared memory

Inter-Process Communication (IPC) is a fundamental concept in operating systems that allows multiple processes to communicate and synchronize their actions.
Inter-Process Communication (IPC) refers to the set of techniques that allow the processes to exchange data and signals with one another.

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.

What is Shared Memory?
The Shared memory is a memory segment that multiple processes can access concurrently. It is one of the fastest IPC methods because the processes communicate by the reading and writing to a shared block of the memory.



How Shared Memory IPC Works?

The Shared memory IPC works by creating a memory segment that is accessible by the multiple processes. Heres a basic outline of how it operates:

Creation of Shared Memory Segment: A process usually the parent, creates a shared memory segment using the system calls like shmget() in Unix-like systems. This segment is assigned the unique identifier (shmid).

Attaching to the Shared Memory Segment: The Processes that need to access the shared memory attach themselves to this segment using shmat() system call. Once attached the processes can directly read from and write to the shared memory.

Synchronization: Process Synchronization is used in a computer system to ensure that multiple processes or threads can run concurrently without interfering with each other.

Detaching and Deleting the Segment: When a process no longer needs access to the shared memory it can detach from the segment using shmdt() system call. The shared memory segment can be removed entirely from system using shmctl() once all processes have the detached.

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.

Writer Process: Writes data into shared memory.
Reader Process: Reads data from shared memory.

Shared memory allows multiple processes to access the same memory segment.

It is **faster** than other IPC mechanisms like message queues or pipes.

The `ftok()` function generates a unique key.

The `shmget()` function creates or retrieves a shared memory segment.

The `shmat()` function attaches the shared memory segment to a process's address space.

The `shmdt()` function detaches the shared memory segment.

The `shmctl()` function can be used to **delete** the shared memory after use.

The **writer** writes data into memory, and the **reader** reads & deletes it.

`ftok("shmfile", 65);`

`ftok()` (File to Key) is a system call that generates a unique key for **Inter-Process Communication (IPC)** mechanisms such as:

- Shared memory (`shmget()`)
- Message queues (`msgget()`)
- Semaphores (`semget()`)

The function syntax is:

```
key_t ftok(const char *pathname, int proj_id);
```

Where:

- `pathname`: A valid file path (e.g., `"shmfile"`) that **must exist**.
- `proj_id`: A small integer (usually between 0 and 255) to differentiate keys for different resources.

```
key_t key = ftok("shmfile", 65);
```

`ftok("shmfile", 65);` generates a **unique key** based on the file `"shmfile"` and the integer `65`.

This key is used by multiple processes to refer to the **same shared memory segment**.

```
int shmid = shmget(key, 1024, 0666 | IPC_CREAT);
```

- The number **0666** is written in **octal** (base 8).
- It defines **read and write permissions** for the **owner, group, and others**.

Octal	Binary	Permission
0	000	No special bits (SetUID, SetGID, Sticky bit)
6	110	Read (r) and Write (w) permissions (No execute x)
6	110	Read (r) and Write (w) permissions (No execute x)
6	110	Read (r) and Write (w) permissions (No execute x)

Thus, **0666 means:**

- **Owner** → Read (`r`) and Write (`w`)
- **Group** → Read (`r`) and Write (`w`)
- **Others** → Read (`r`) and Write (`w`)

It allows any user to read and write to the shared memory segment, but not execute.

```
char *str = (char*) shmat(shmid, (void*)0, 0);
```

`shmat()` attaches the shared memory segment to the **process's address space**.
`shmid`: The shared memory identifier.
`(void*)0`: Allows the OS to **choose** the address for attachment.
`0`: No special flags.

`shmdt(str);` **detaches** the shared memory segment from the process.
This **does NOT delete** the shared memory, only removes access.

Task 4

Task 4: To write a C program to implement concept of Shared memory

Program:

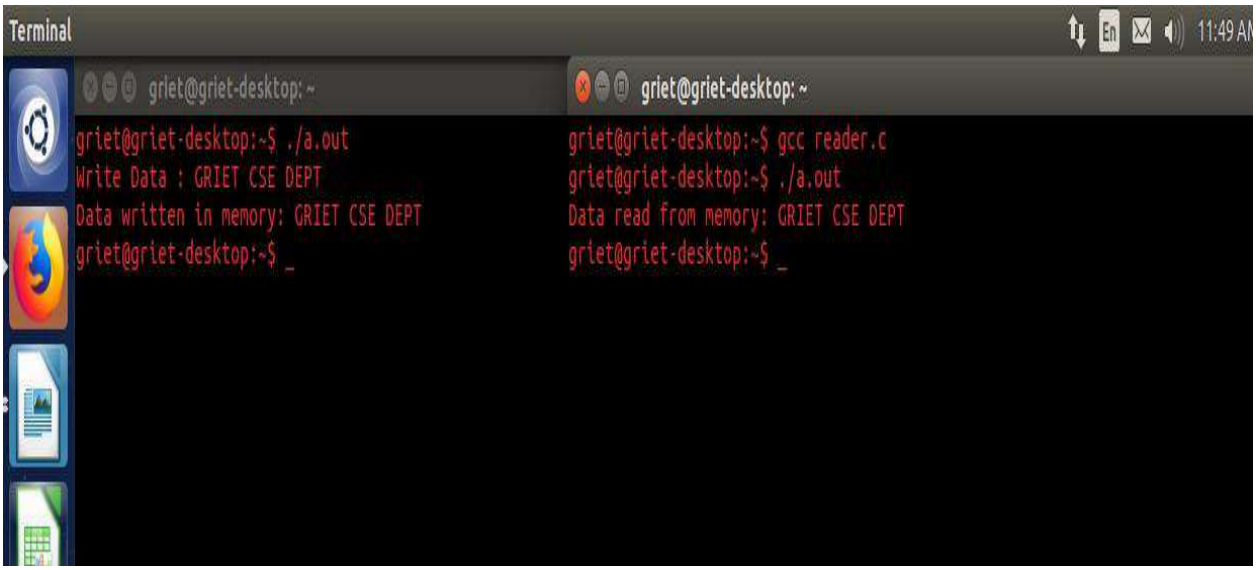
Writer

```
#include <sys/ipc.h>
 #include <sys/shm.h>
#include <stdio.h>
#include <string.h>
 int main()
{
// ftok to generate unique key
 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("Write Data : ");
gets(str);
printf("Data written in memory: %s\n",str);
//detach from shared memory
shmdt(str);
return 0;
}
```

Reader

```
#include <sys/ipc.h>
 #include <sys/shm.h>
#include <stdio.h>
int main()
{
// ftok to generate unique key
 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;
}
```

Output:


Terminal
11:49 AM
griet@griet-desktop: ~
griet@griet-desktop:~$ ./a.out
Write Data : GRIET CSE DEPT
Data written in memory: GRIET CSE DEPT
griet@griet-desktop:~$ _
griet@griet-desktop: ~
griet@griet-desktop:~$ gcc reader.c
griet@griet-desktop:~$ ./a.out
Data read from memory: GRIET CSE DEPT
griet@griet-desktop:~$ _