# SHARED MEMORY

Asst.Prof Rachna Karnavat,PICT IT

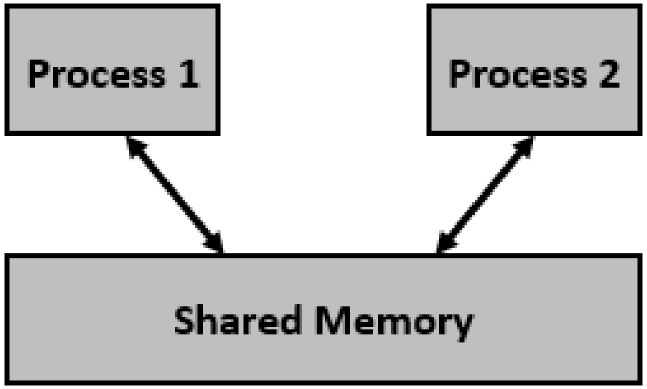
## What is shared memory?

Answer::

Shared memory is **the fastest interprocess communication mechanism**.

The operating system maps a memory segment in the address space of several processes, so that several processes can read and write in that memory segment without calling operating system functions.

Inter Process Communication through shared memory is a concept where two or more process can access the common memory



Inter Process Communication through shared memory is a concept where two or more process 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 process 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 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 input file into shared memory and from shared memory to the output file.

## System Calls for Shared Memory

- Create the shared memory segment or use an already created shared memory segment (shmget())
- Attach the process to the already created shared memory segment (shmat())
- Detach the process from the already attached shared memory segment (shmdt())
- Control operations on the shared memory segment (shmctl())

## Header Files To Be Included:

```
#include <iostream>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <stdio.h>
using namespace std;
```

***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);*

## key_t ftok(const char *path, int id);

- Path - (Input) The path name of the file used in combination with id to generate the key.

- Id - (Input) The integer identifier used in combination with path to generate the key.

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

- The first argument, key, recognizes the shared memory segment. It is derived from the library function ftok().
- The second argument, size, is the size of the shared memory segment rounded to multiple of PAGE_SIZE.
- The third argument, shmflg, specifies the required shared memory flag/s such as IPC_CREAT (creating new segment) or IPC_EXCL (Used with IPC_CREAT to create new segment and the call fails, if the segment already exists).
- This call returns a valid shared memory identifier (used for further calls of shared memory) on success or -1 in case of failure.

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

- The first argument, shmid, is the identifier of the shared memory segment. This id is the shared memory identifier, which is the return value of shmget() system call.

- The second argument, shmaddr, is to specify the attaching address. If shmaddr is NULL, the system by default chooses the suitable address to attach the segment.

- The third argument, shmflg, specifies the required shared memory flag/s.

- This call would return the address of attached shared memory segment on success and -1 in case of failure.

## int shmdt(const void *shmaddr)

- The argument, shmaddr, is the address of shared memory segment to be detached. The to-be-detached segment must be the address returned by the shmat() system call.
- This call would return 0 on success and -1 in case of failure.

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

- The second argument, cmd, is the command to perform the required control operation on the shared memory segment.

  Valid values for cmd are −

  - IPC_STAT − Copies the information of the current values of each member of struct shmid_ds to the passed structure pointed by buf. This command requires read permission to the shared memory segment.
  - IPC_SET − Sets the user ID, group ID of the owner, permissions, etc. pointed to by structure buf.
  - IPC_RMID − Marks the segment to be destroyed. The segment is destroyed only after the last process has detached it.
  - IPC_INFO − Returns the information about the shared memory limits and parameters in the structure pointed by buf.
  - SHM_INFO − Returns a shm_info structure containing information about the consumed system resources by the shared memory.

- The third argument, buf, is a pointer to the shared memory structure named struct shmid_ds. The values of this structure would be used for either set or get as per cmd.

- Upon success of IPC_INFO and SHM_INFO or SHM_STAT returns the index or identifier of the shared memory segment or 0 for other operations and -1 in case of failure.

## shmid_ds

```
struct shmid_ds {
    struct ipc_perm shm_perm;    /* Ownership and permissions */
    size_t          shm_segsz;   /* Size of segment (bytes) */
    time_t          shm_atime;   /* Last attach time */
    time_t          shm_dtime;   /* Last detach time */
    time_t          shm_ctime;   /* Last change time */
    pid_t           shm_cpid;    /* PID of creator */
    pid_t           shm_lpid;    /* PID of last shmat()/shmdt() */
    shmatt_t        shm_nattch;  /* No. of current attaches */
    ...
};

struct ipc_perm {
    key_t key;            /* Key supplied to shmget() */
    uid_t uid;            /* Effective UID of owner */
    gid_t gid;            /* Effective GID of owner */
    uid_t cuid;           /* Effective UID of creator */
    gid_t cgid;           /* Effective GID of creator */
    unsigned short mode;  /* Permissions + SHM_DEST and
                             SHM_LOCKED flags */
    unsigned short seq;   /* Sequence number */
};
```

# Code

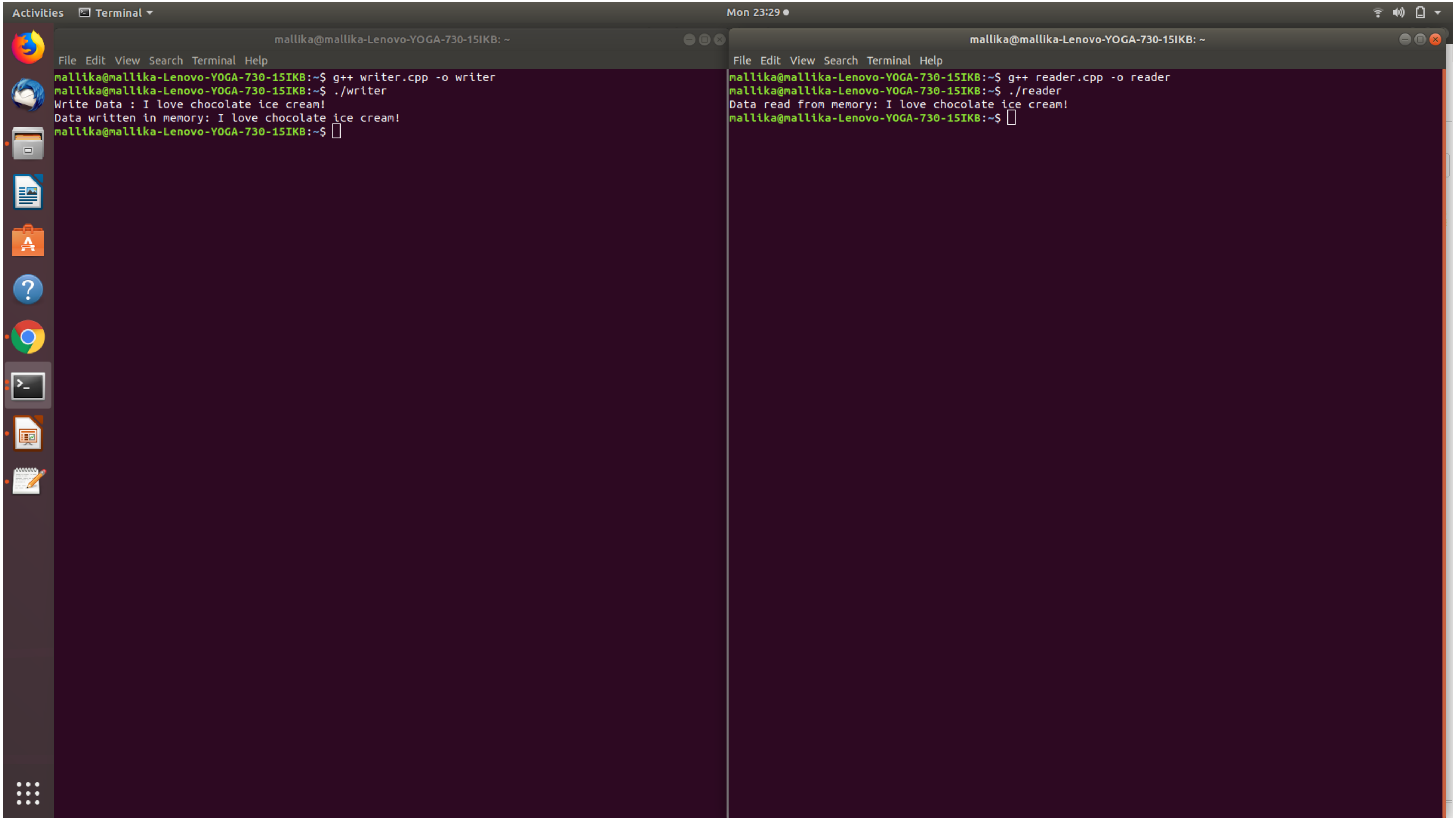
WRITER

```
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);

      cout<<"Write Data : ";

    cin.getline(str, 1024);

      printf("Data written in memory: %s\n",str);

          //detach from shared memory

    shmdt(str);

      return 0;

}
```

READER

```
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


Activities
Terminal
Mon 23:29
mallika@mallika-Lenovo-YOGA-730-15IKB: ~
File Edit View Search Terminal Help
mallika@mallika-Lenovo-YOGA-730-15IKB: ~
File Edit View Search Terminal Help

```
mallika@mallika-Lenovo-YOGA-730-15IKB:~$ g++ writer.cpp -o writer
mallika@mallika-Lenovo-YOGA-730-15IKB:~$ ./writer
Write Data : I love chocolate ice cream!
Data written in memory: I love chocolate ice cream!
mallika@mallika-Lenovo-YOGA-730-15IKB:~$
```

```
mallika@mallika-Lenovo-YOGA-730-15IKB:~$ g++ reader.cpp -o reader
mallika@mallika-Lenovo-YOGA-730-15IKB:~$ ./reader
Data read from memory: I love chocolate ice cream!
mallika@mallika-Lenovo-YOGA-730-15IKB:~$
```

THANK YOU FOR
LISTENING TO OUR PRESENTATION
memegenerator.net