Operating System Tutorial

Tutorial-2

Inter Process Communication(IPC)

(Shared Memory)

Objectives

- To create a Shared Memory Segment
- To Attach and Detach a Shared Memory Segment
- To control a Shared Memory Segment
- To show how the processes can communicate among themselves using the Shared Memory regions.

IPC (Inter-Process Communication)

- Inter process communication (IPC) is a mechanism which allows processes to communicate each other and synchronize their actions.
- The communication between these processes can be seen as a method of co-operation between them.
- A process can be of two type:
  - Independent process (Unrelated).
  - Co-operating process (Related).

Types of IPC

- Among related processes on same system (PIPE, Shared Memory with IPC_PRIVATE option).
- Among unrelated processes on same system (FIFO, Shared Memory with given key).
- Among unrelated processes on physically different systems (SOCKET).

What is Shared Memory ?

- Shared Memory is an efficeint means of passing data between programs. One program will create a memory portion which other processes (if permitted) can access.

How it Works?

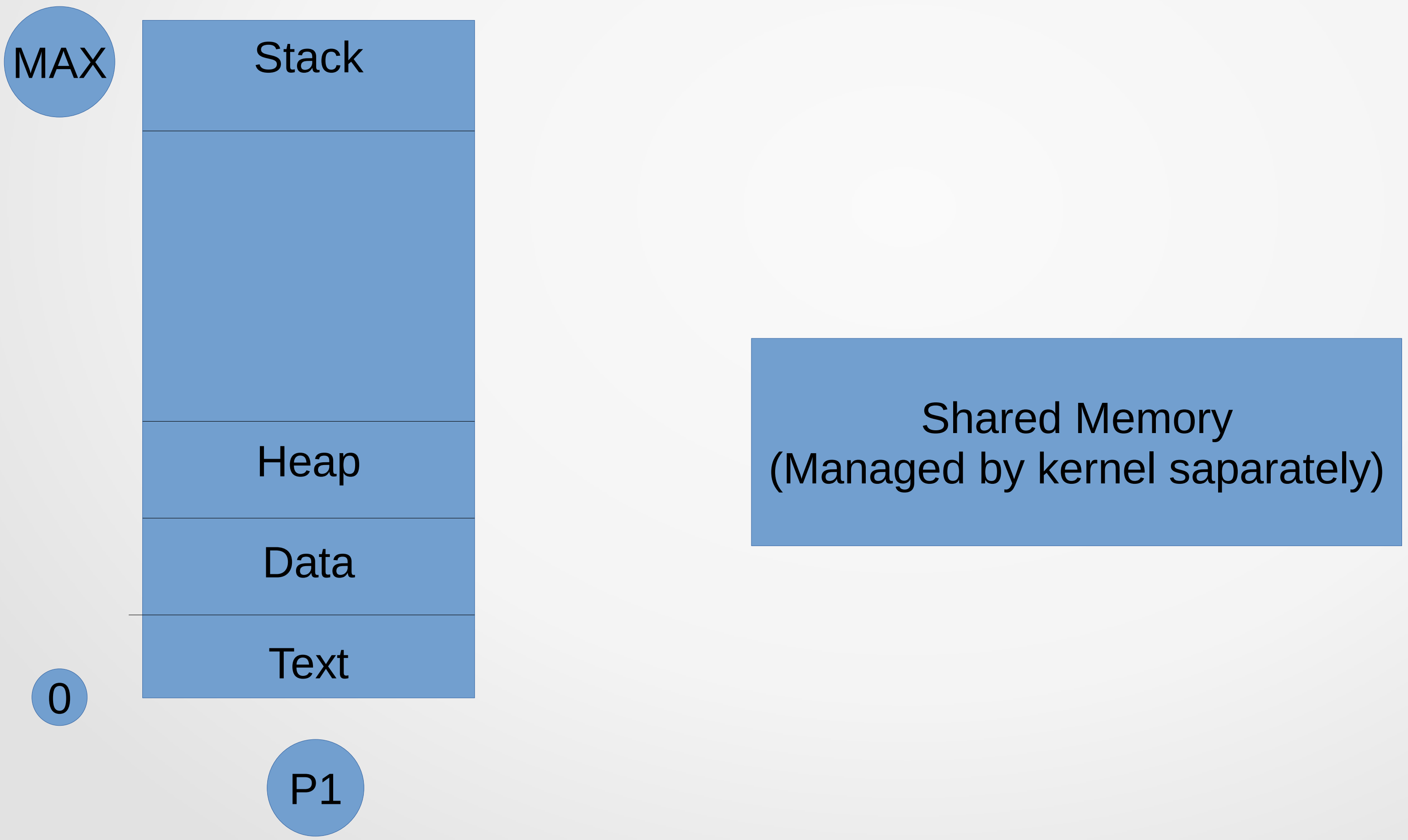
- A process has virtual address space with sections like “text”, “data”, “heap”, “stack” etc.
- This virtual address is same for all processes in OS.
- On 64-bit system this virtual address ranges from 00400000-ffffffffff601000.
- This virtual address space maps into physical address space.

How it Works?

- A process creates a memory segment (at virtual level only), just like it creates a file.
- Initially this memory segment is not attached to creater process. This segement do not have a address space.
- This is a seperate section of memory from creater's virtual address space. And managed by kernel only (not by creater process).
- Creater process can later attach this memory segment to his address space .
- The address where this segment is going to attach can be decided by programmer, or kernel.

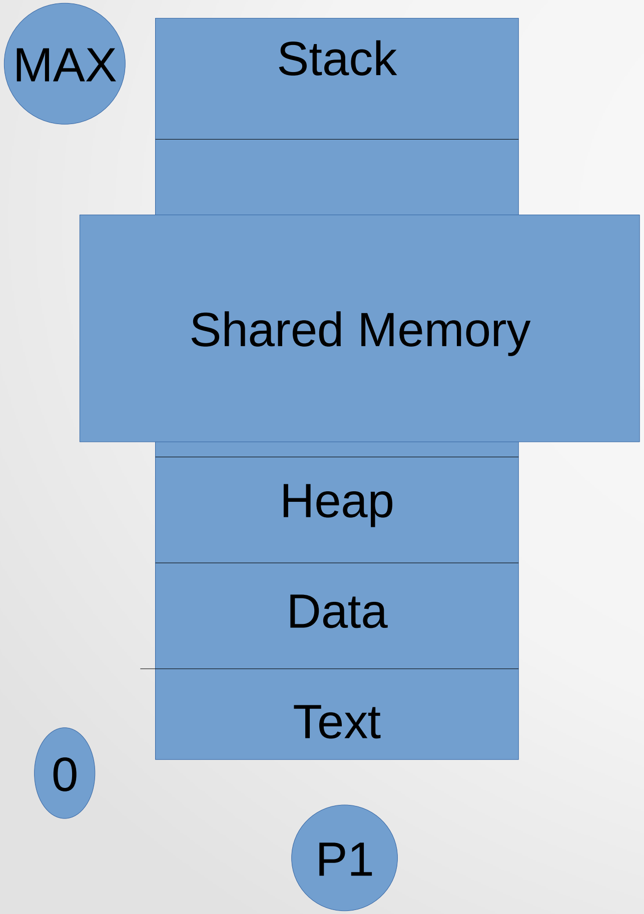
How it Works?

- Process P1 creates shared memory. Which is seperate from P1's address space.



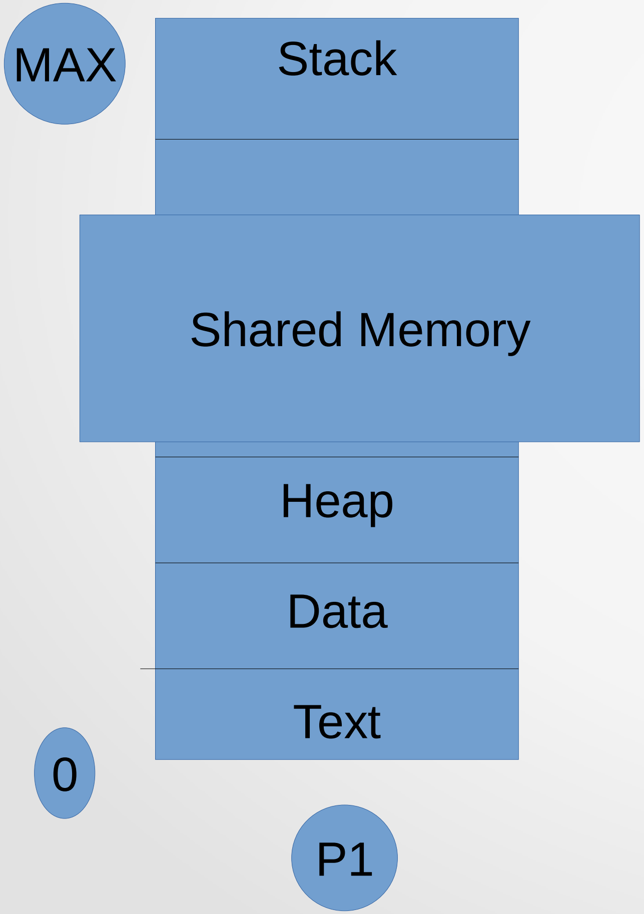
How it Works?

- Process P1 can attache this Shared Memory to its virtual address space later on given or default address.



How it Works?

- This shared memory can be attached at address **<** ***00400000 (0)*** or **>** ***ffffffffff601000 (MAX)*** or ***inbetween***.

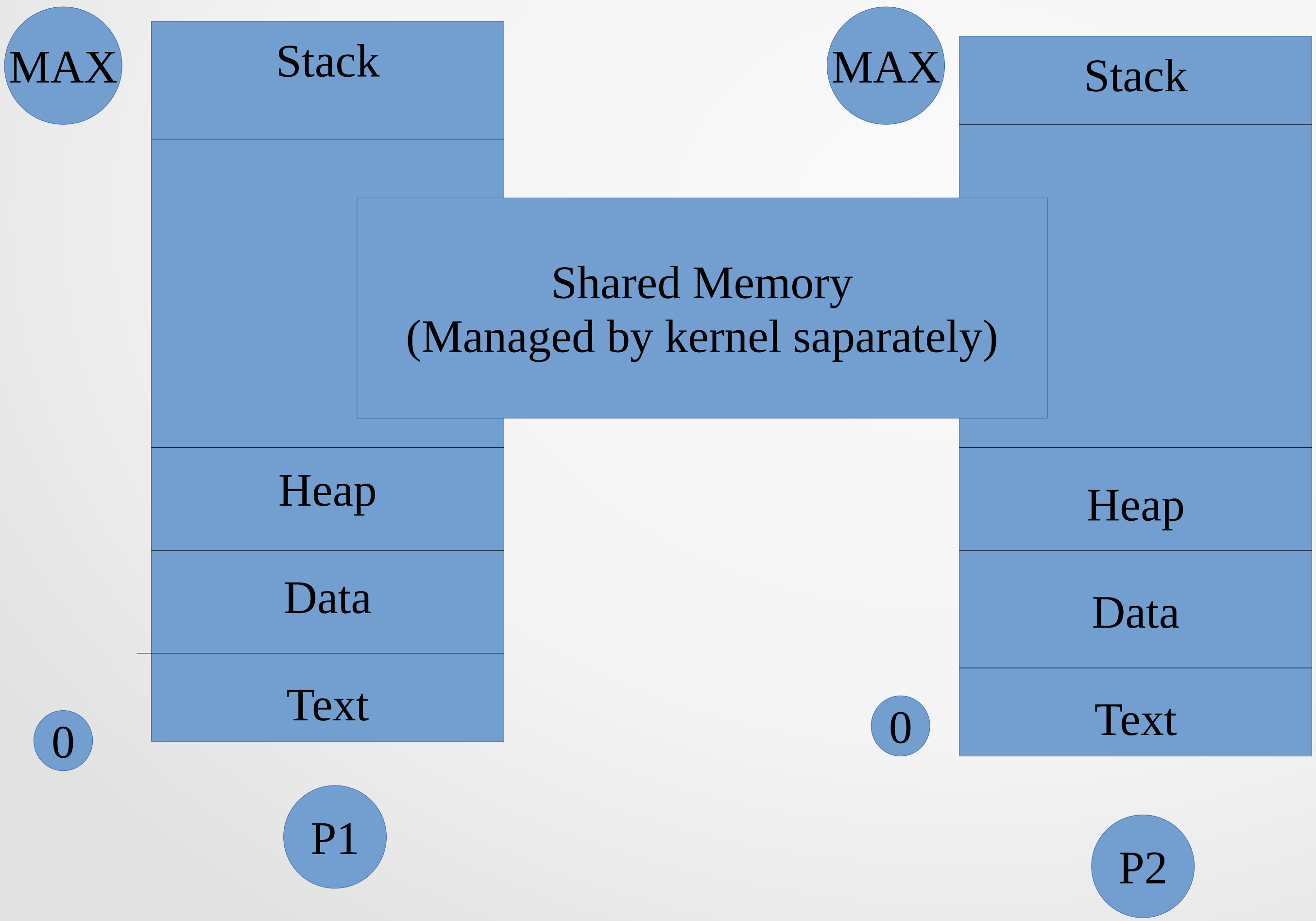


IPC with Shared Memory

- Once shared memory has been created, kernel takes care of shared memory.
- We can see all shared memories by *ipcs* command..
- These shared memories are created with some initial permissions.
- Based on permissions other processes can also attache this shared memory and use it.
- If two processes attache same shared memory then both can communicate. One write to that memory and other can read.

IPC with Shared Memory

- Process P1 created and attached shared memory. Process P2 just attached it. Now both can communicate.



Shmget(key, size, permission) System Call

- Creates a new Shared Memory.
- Key:
  - It is a 32-bit integer.
  - It is key of shared memory. Other processes uses shared memory using this key.
  - Any 32-bit integer can be used.
  - IPC_PRIVATE can be used. Its value is zero. This key makes shared memory only usable for related processes.
  - We can use *ftok()* system call to generate gurantee unique key.

*key_t ftok(const char *path, int id)* System Call

- The ftok() function shall return a key based on path and id.
- The ftok() function shall return the same key value for all paths that name the same file, when called with the same id value.
- Return different key values when called with different id values or with paths that name different files existing on the same file system at the same time.
- *key_t* present in *sys/ipc.h*. We can use *int* instead.
- Upon successful completion, ftok() shall return a key.
- Upon unsuccessful completion, ftok() shall return -1.

Shmget(key, size, permission) System Call

- size:
  - Specifies the size in bytes of the shared memory segment.

Shmget(key, size, permission) System Call

- Permission:
  - IPC_CREAT: Create a shared memory segment if a shared memory identifier does not exist for the specified key parameter. IPC_CREAT is ignored when IPC_PRIVATE is specified for the key parameter.
  - IPC_EXCL.
  - TPF_IPC64.
  - S_IRUSR
  - S_IWUSR
  - S_IRGRP
  - S_IWGRP
  - S_IROTH
  - S_IWOTH

Shmget(key, size, permission) System Call

- Returns segment_id managed by kernel for given shared memory segment.
- Return -1 if fails.
- Fails when try to create two shared memoies with same key.

- segment_id = shmget (key, shared_segment_size, IPC_CREAT | IPC_EXCL | S_IRUSR | S_IWUSR);

Shmat(segment_id, address, mode)

- Attach given shared memory on process virtual address space.
- segment_id:
  - Value returned by shmget().
  - You can use any *shmid* from output of “*ipcs*”.
- address:
  - Where to attach given segment.
  - You can give your own address.
  - If NULL then kernel will decide where to attach.
- Mode:
  - Read or write mode of attachement.
  - 0: for both read and write
- Returns attached segment address as a void pointer.

shared memory using command

- *ipcs*: see all IPC sections
- *ipcrm shm segment_id* : delete given segment.
- *ipcmk -M size*: Create Shared memory of given size.

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

- performs the control operation specified by *cmd* on the shared memory segment whose identifier is given in *shmid*.

- With two structures.

```
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(2)/shmdt(2) */
    shmatt_t        shm_nattch;  /* No. of current attaches */
    ...
};
```

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

- Second structure used by first one:

```
struct ipc_perm {
    key_t          __key;    /* Key supplied to shmget(2) */
    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 */
};;
```

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

- shmid is shared memory ID.
- Create a variable of type shmid_ds. Pass its address in place of buf.
- Commands(cmd):
  - IPC_STAT: Copy information from the kernel data structure.
  - IPC_SET: Write the values of some members of the shmid_ds structure pointed to by buf.
  - IPC_RMID
  - ...

Detach the shared memory segment

- shmdt (value returned by shmat);
- Only detach from process.
- Shared memory is still exists.

Deallocate the shared memory

- shmctl (segment_id, IPC_RMID, 0);
- Delete shared memory from system.

Interprocess Communication

- Among related processes all address space will be copied. Thats how attached shared memory also gets copied.
- Among unrelated processes we have to use same key both client and server side.

General Scheme for IPC using SM (Producer)

1) Ask for a shared memory with a memory key and memorize the returned shared memory ID. This is performed by system call shmget( ).
2) Attach this shared memory to the server's address space with system call shmat( ).
3) Initialize the shared memory, if necessary.
4) Do something and wait for consumer(s) for completion.
5) Detach the shared memory with system call shmdt( ).
6) Remove the shared memory with system call shmctl( ).

General Scheme for IPC using SM (Consumer)

1) Ask for a shared memory with the same memory key and memorize the returned shared memory ID.
2) Attach this shared memory to the it’s address space.
3) Use the shared memory.
4) Detach all shared memory segments, if necessary.
5) Exit.

Example

Write a C program to demonstrate IPC (Shared memory) using two processes named Reader and Writer. Reader process will write data on shared memory and Writer process will read (and display) the data written by earlier process.

Lab Excercise

Task 1

Modify the previous program to do following:

Writer process creates an array of Prime Numbers. Reader reads the prime number by passing the index of it.

Task 2

Write a program in C to create a shared memory, and print its properties like:

Size of segment, Last attach time, Last detach time, Last change time, PID of creator , No. of current attaches, Effective UID, Effective GID, Sequence Number and Key.

Task 3

Write a program in C where server pass 4 numbers to client and client add those numbers. Both Server and Client are unrelated processes.

Task 4

Repeat above program as IPC_PRIVATE as key. And values are passing between parent and child.

Task 5

A program that creates a shared memory segment and waits until two other separate processes writes something into that shared memory segment after which it prints what is written in shared memory.

Process 1 writes:	Hello
Process 2 writes:	Hi
Process 3 writes:	Hii
Process 1 Prints:	Everything written on shared memory