Shared Memory Implementation in Linux

Tushar B. Kute,
http://tusharkute.com

Shared Memory?

- It allows two unrelated processes to access the same logical memory.
- Shared memory is a very efficient way of transferring data between two running processes.
- Although the X/Open standard doesn't require it, it's probable that most implementations of shared memory arrange for the memory being shared between different processes to be the same physical memory.

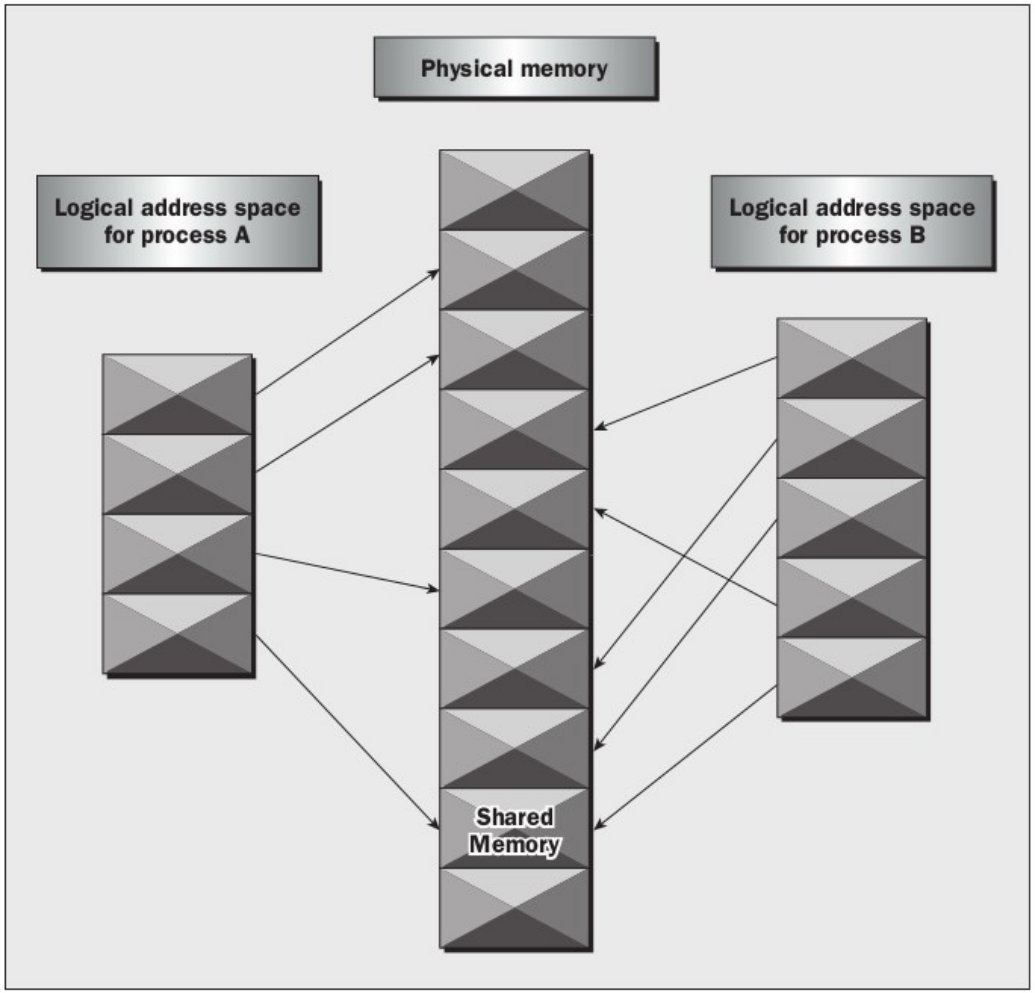
Shared Memory?

- Shared memory is a special range of addresses that is created by IPC for one process and appears in the address space of that process.
- Other processes can then “attach” the same shared memory segment into their own address space. All processes can access the memory locations just as if the memory had been allocated by malloc .
- If one process writes to the shared memory, the changes immediately become visible to any other process that has access to the same shared memory.

Synchronization in Shared Memory?

- Shared memory provides an efficient way of sharing and passing data between multiple processes.
- By itself, shared memory doesn’t provide any synchronization facilities. Because it provides no synchronization facilities, you usually need to use some other mechanism to synchronize access to the shared memory.
- Typically, you might use shared memory to provide efficient access to large areas of memory and pass small messages to synchronize access to that memory.

Shared Memory



The functions

- The functions for shared memory resemble those for semaphores:

```
#include <sys/shm.h>
void *shmat(int shm_id, const void *shm_addr, int shmflg);
int shmctl(int shm_id, int cmd, struct shmid_ds *buf);
int shmdt(const void *shm_addr);
int shmget(key_t key, size_t size, int shmflg);
```

- As with semaphores, the include files sys/types.h and sys/ipc.h are normally automatically included by shm.h .

shmget

- You create shared memory using the shmget function:

  int shmget(key_t key, size_t size, int shmflg);
- As with semaphores, the program provides key , which effectively names the shared memory segment, and the shmget function returns a shared memory identifier that is used in subsequent shared memory functions. There's a special key value, IPC_PRIVATE , that creates shared memory private to the process.
- The second parameter, size , specifies the amount of memory required in bytes.
- The third parameter, shmflg , consists of nine permission flags that are used in the same way as the mode flags for creating files. A special bit defined by IPC_CREAT must be bitwise ORed with the permissions to create a new shared memory segment.

shmat

- When you first create a shared memory segment, it’s not accessible by any process. To enable access to the shared memory, you must attach it to the address space of a process. You do this with the shmat function:

  void *shmat(int shm_id, const void *shm_addr, int shmflg);
- The first parameter, shm_id , is the shared memory identifier returned from shmget .
- The second parameter, shm_addr , is the address at which the shared memory is to be attached to the current process. This should almost always be a null pointer, which allows the system to choose the address at which the memory appears.
- The third parameter, shmflg, is a set of bitwise flags. The two possible values are SHM_RND, which, in conjunction with shm_addr , controls the address at which the shared memory is attached, and SHM_RDONLY , which makes the attached memory read-only.

shmctl

- The control functions for shared memory are (thankfully) somewhat simpler than the more complex ones for semaphores:

  int shmctl(int shm_id, int command, struct shmid_ds *buf);
- The shmid_ds structure has at least the following members:

```
struct shmid_ds {
uid_t shm_perm.uid;
uid_t shm_perm.gid;
mode_t shm_perm.mode;
}
```

- The first parameter, shm_id , is the identifier returned from shmget .

shmctl

- The second parameter, command , is the action to take. It can take three values, shown in the following table.

Command	Description
IPC_STAT	Sets the data in the shmid_ds structure to reflect the values associated with the shared memory.
IPC_SET	Sets the values associated with the shared memory to those provided in the shmid_ds data structure, if the process has permission to do so.
IPC_RMID	Deletes the shared memory segment.

- The third parameter, buf , is a pointer to the structure containing the modes and permissions for the shared memory.

Programs

- Lets do the coding now...

Thank you

This presentation is created using LibreOffice Impress 5.3.2.2, can be used freely as per GNU General Public License

@mitu_skillologies

/mITuSkillologies

@mitu_group

Web Resources
http://mitu.co.in
http://tusharkute.com

Blogs
http://digitallocha.blogspot.in
http://kyamputar.blogspot.in

tushar@tusharkute.com