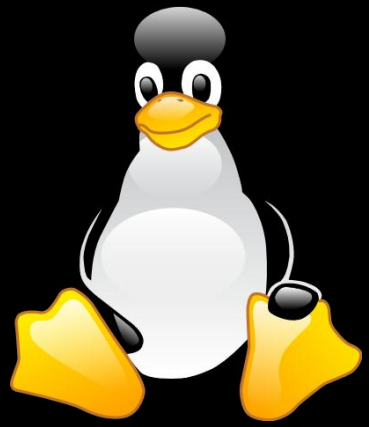
Tushar B Kute,
Assistant Professor,
Sandip Institute of Technology and Research Centre, Nashik (INDIA)
tbkute@gmail.com

University of Pune
T.E. I.T.
Subject code: 314441

OPERATING SYSTEM

Part 16 : Semaphore And Shared Memory



Semaphore Function Definitions

```
#include <sys/sem.h>

int semctl(int sem_id, int sem_num, int command, ...);
int semget(key_t key, int num_sems, int sem_flags);
int semop(int sem_id, struct sembuf *sem_ops, size_t
    num_sem_ops);
```

Semaphore

- The header file sys/sem.h usually relies on two other header files, sys/types.h and sys/ipc.h. Normally they are automatically included by sys/sem.h and you do not need to explicitly add a #include for them.
- As we work through each function in turn, remember that these functions were designed to work for arrays of semaphore values, which makes their operation significantly more complex than would have been required for a single semaphore.

semget

- The semget function creates a new semaphore or obtains the semaphore key of an existing semaphore:

  int semget(key_t key, int num_sems, int sem_flags);

- The first parameter, *key*, is an integral value used to allow unrelated processes to access the same semaphore.
- All semaphores are accessed indirectly by the program supplying a key, for which the system then generates a semaphore identifier. The semaphore key is used only with semget. All other semaphore functions use the semaphore identifier returned from semget.

semget

- There's a special key value, IPC_PRIVATE, that creates semaphore private to the process.
- The num_sems parameter is the number of semaphores required. This is almost always 1.
- The sem_flags parameter is a set of flags, very much like the flags to the open function.
- The semget function returns a positive (nonzero) value on success; this is the semaphore identifier used in the other semaphore functions. On error, it returns –1.

semop

- The function semop is used for changing the value of the semaphore:

  int semop(int sem_id, struct sembuf *sem_ops, size_t num_sem_ops);

- The first parameter, sem_id, is the semaphore identifier, as returned from semget. The second parameter, sem_ops, is a pointer to an array of structures, each of which will have at least the following members:

```
struct sembuf
 {
    short sem_num;
    short sem_op;
    short sem_flg;
 }
```

semop

- The first member, sem_num, is the semaphore number, usually 0 unless you're working with an array of semaphores. The sem_op member is the value by which the semaphore should be changed. Only two values are allowed i.e. 1 and -1
- The final member, sem_flg, is usually set to 0.

semctl

- The semctl function allows direct control of semaphore information:

  int semctl(int sem_id, int sem_num, int command, ...);

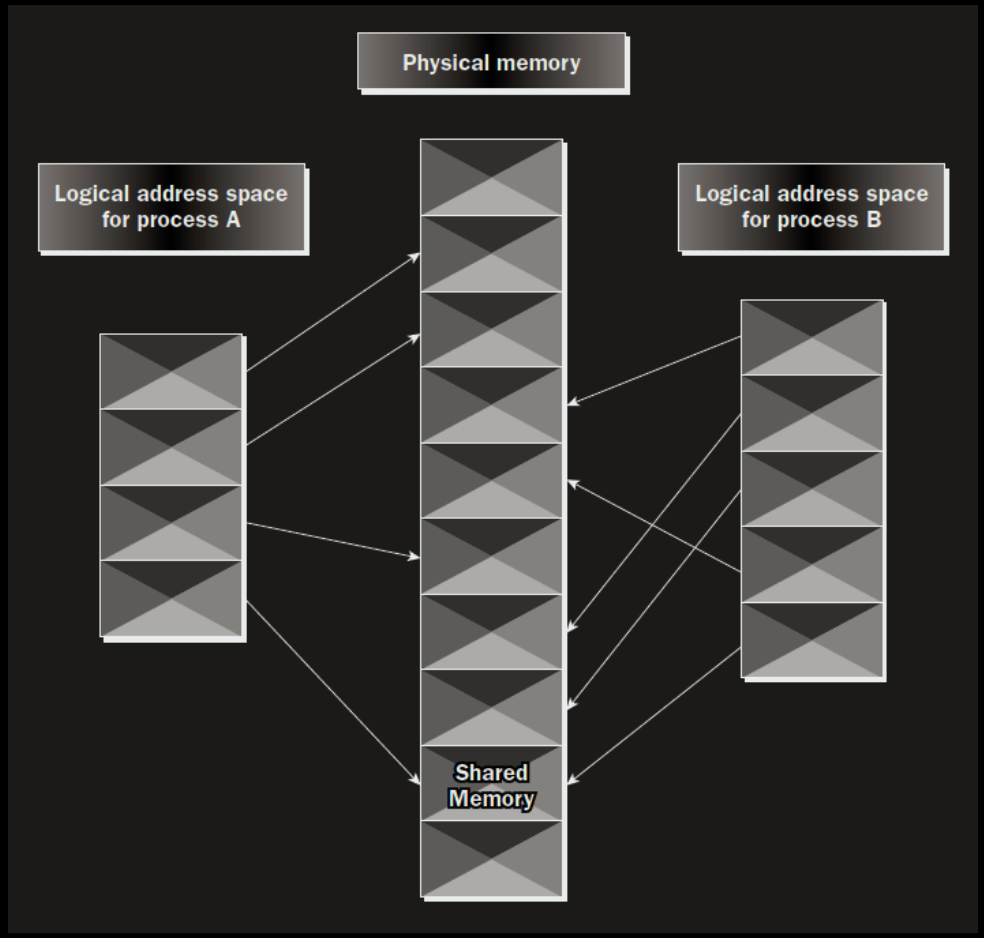
- The first parameter, sem_id, is a semaphore identifier, obtained from semget. The sem_num parameter is the semaphore number. You use this when you're working with arrays of semaphores. Usually, this is 0, the first and only semaphore.
- The command parameter is the action to take, and a fourth parameter, if present, is a union semun, which according to the X/OPEN specification must have at least the following members:

```
union semun
{
    int val;
    struct semid_ds *buf;
    unsigned short *array;
}
```

semctl

- The two common values of command are:
- SETVAL: Used for initializing a semaphore to a known value. The value required is passed as the val member of the union semun. This is required to set the semaphore up before it's used for the first time.
- IPC_RMID: Used for deleting a semaphore identifier when it's no longer required.

Shared Memory



Functions

- **#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. It's not an error to have the IPC_CREAT flag set and pass the key of an existing shared memory segment. The IPC_CREAT flag is silently ignored if it is not required.

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);

shmat

- 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_R and SHM_W.

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;
}
```

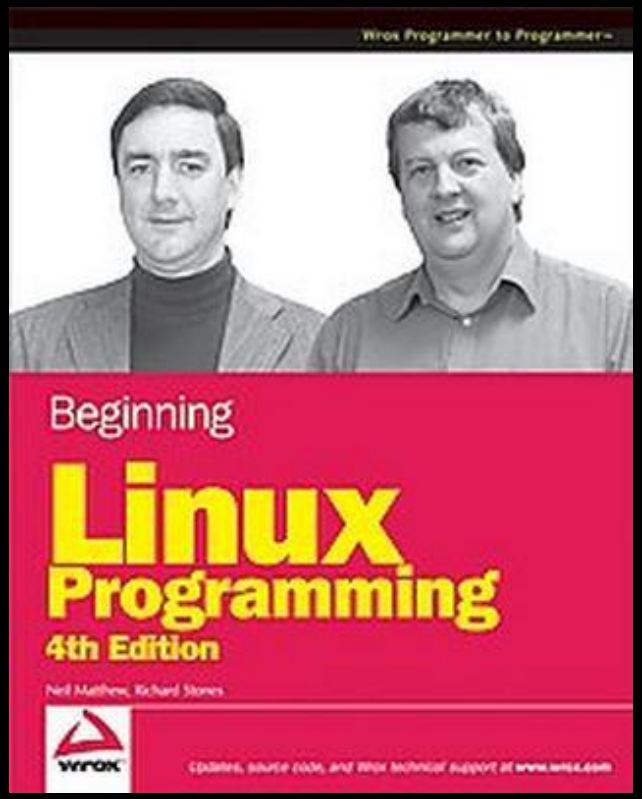
shmctl

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

Reference Books



- "Beginning Linux Programming", 4th Edition, Neil Mathew, Richard Stones, Wrox Publication.
- Rating: