ASSIGNMENT NO: 7 (B)

AIM: Inter process communication in Linux using Shared Memory

Shared Memory

Shared memory can best be described as the mapping of an area (segment) of memory that will be mapped and shared by more than one process. This is by far the fastest form of IPC, because there is no intermediation (i.e. a pipe, a message queue, etc). Instead, information is mapped directly from a memory segment, and into the addressing space of the calling process. A segment can be created by one process, and subsequently written to and read from by any number of processes.
SYSTEM CALL: shmget()
In order to create a new message queue, or access an existing queue, the shmget() system call is used.

```
SYSTEM CALL: shmget();

PROTOTYPE: int shmget ( key_t key, int size, int shmflg );
  RETURNS: shared memory segment identifier on success
        -1 on error: errno = EINVAL (Invalid segment size specified)
                    EEXIST (Segment exists, cannot create)
                    EIDRM (Segment is marked for deletion, or was removed)
                    ENOENT (Segment does not exist)
                    EACCES (Permission denied)
                    ENOMEM (Not enough memory to create segment)
NOTES:
```

This particular call should almost seem like old news at this point. It is strikingly similar to the corresponding get calls for message queues and semaphore sets.

The first argument to shmget() is the key value (in our case returned by a call to ftok()). This key value is then compared to existing key values that exist within the kernel for other shared memory segments. At that point, the open or access operation is dependent upon the contents of the shmflg argument.

IPC_CREAT

Create the segment if it doesn't already exist in the kernel.

IPC_EXCL

When used with IPC_CREAT, fail if segment already exists.

If IPC_CREAT is used alone, shmget() either returns the segment identifier for a newly created segment, or returns the identifier for a segment which exists with the same key value. If IPC_EXCL is used along with IPC_CREAT, then either a new segment is created, or if the segment exists, the call fails with -1. IPC_EXCL is useless by itself, but when combined with IPC_CREAT, it can be used as a facility to guarantee that no existing segment is opened for access.

Once again, an optional octal mode may be OR'd into the mask.

Let's create a wrapper function for locating or creating a shared memory segment :

```
int open_segment( key_t keyval, int segsize )
{
        int     shmid;

        if((shmid = shmget( keyval, segsize, IPC_CREAT | 0660 )) == -1)
        {
                return(-1);
        }

        return(shmid);
}
```

Note the use of the explicit permissions of 0660. This small function either returns a shared memory segment identifier (int), or -1 on error. The key value and requested segment size (in bytes) are passed as arguments.

Once a process has a valid IPC identifier for a given segment, the next step is for the process to attach or map the segment into its own addressing space.

SYSTEM CALL: shmat()

```
  SYSTEM CALL: shmat();

  PROTOTYPE: int shmat ( int shmid, char *shmaddr, int shmflg);
    RETURNS: address at which segment was attached to the process, or
           -1 on error: errno = EINVAL (Invalid IPC ID value or attach address passed)
                                ENOMEM (Not enough memory to attach segment)
                                EACCES (Permission denied)
  NOTES:
```

If the addr argument is zero (0), the kernel tries to find an unmapped region. This is the recommended method. An address can be specified, but is typically only used to facilitate proprietary hardware or to resolve conflicts with other apps. The SHM_RND flag can be OR'd into the flag argument to force a passed address to be page aligned (rounds down to the nearest page size).

In addition, if the SHM_RDONLY flag is OR'd in with the flag argument, then the shared memory segment will be mapped in, but marked as readonly.

This call is perhaps the simplest to use. Consider this wrapper function, which is passed a valid IPC identifier for a segment, and returns the address that the segment was attached to:

```
char *attach_segment( int shmid )
{
        return(shmat(shmid, 0, 0));
}
```

Once a segment has been properly attached, and a process has a pointer to the start of that segment, reading and writing to the segment become as easy as simply referencing or dereferencing the pointer! Be careful not to lose the value of the original pointer! If this happens, you will have no way of accessing the base (start) of the segment.

SYSTEM CALL: shmctl()

```
  SYSTEM CALL: shmctl();
  PROTOTYPE: int shmctl ( int shmqid, int cmd, struct shmid_ds *buf );
    RETURNS: 0 on success
           -1 on error: errno = EACCES (No read permission and cmd is IPC_STAT)
                          EFAULT (Address pointed to by buf is invalid with IPC_SET and
                                  IPC_STAT commands)
                          EIDRM  (Segment was removed during retrieval)
                          EINVAL (shmqid invalid)
                          EPERM  (IPC_SET or IPC_RMID command was issued, but
                                  calling process does not have write (alter)
                                  access to the segment)
     NOTES:
```

This particular call is modeled directly after the *msgctl* call for message queues. In light of this fact, it won't be discussed in too much detail. Valid command values are:

IPC_STAT

Retrieves the shmid_ds structure for a segment, and stores it in the address of the buf argument

IPC_SET

Sets the value of the ipc_perm member of the shmid_ds structure for a segment. Takes the values from the buf argument.

IPC_RMID

Marks a segment for removal.

The IPC_RMID command doesn't actually remove a segment from the kernel. Rather, it marks the segment for removal. The actual removal itself occurs when the last process currently attached to the segment has properly detached it. Of course, if no processes are currently attached to the segment, the removal seems immediate.

To properly detach a shared memory segment, a process calls the *shmdt system call.*

SYSTEM CALL: shmdt()

```
 SYSTEM CALL: shmdt();

 PROTOTYPE: int shmdt ( char *shmaddr );
   RETURNS: -1 on error: errno = EINVAL (Invalid attach address passed)
```

After a shared memory segment is no longer needed by a process, it should be detached by calling this system call. As mentioned earlier, this is not the same as removing the segment from the kernel! After a detach is successful, the shm_nattch member of the associates shmid_ds structure is decremented by one. When this value reaches zero (0), the kernel will physically remove the segment.

Conclusion:

Thus we studied IPC mechanisms