Assignment No. 7(B)

Date:

TITLE: Inter-process Communication using Shared Memory using System V. Application to demonstrate: Client and Server Programs in which server process creates a shared memory segment and writes the message to the shared memory segment. Client process reads the message from the shared memory segment and displays it to the screen.

OBJECTIVE:

- Study the concepts Inter Process Communication using Shared Memory in Linux.
- Understand the concept of Client and Server Programming using shared memory.
- Learn to implement Client and Server processes in the Linux kernel.

SOFTWARE REQUIREMENTS:

1. Ubuntu 16.04
2. GNU C Compiler

THEORY:

System V IPC Mechanisms

Linux supports three types of interprocess communication mechanisms which first appeared in Unix System V (1983). These are message queues, semaphores and shared memory. These System V IPC mechanisms all share common authentication methods. Processes may access these resources only by passing a unique reference identifier to the kernel via system calls. Access to these System V IPC objects is checked using access permissions, much like accesses to files are checked. The access rights to the System V IPC object is set by the creator of the object via system calls. The object's reference identifier is used by each mechanism as an index into a table of resources. It is not a straight forward index but requires some manipulation to generate the index.

All Linux data structures representing System V IPC objects in the system include an `ipc_perm` structure which contains the owner and creator processes user and group identifiers. the access mode for this object (owner, group and other) and the IPC object's key. The key is used as a way of locating the System V IPC object's reference identifier. Two sets of key are supported: public and private. If the key is public then any process in the system, subject to rights checking, can find the reference identifier for the System V IPC object. System V IPC objects can never be referenced with a key, only by their reference identifier.

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.

Kernel *shmid_ds* structure

As with message queues and semaphore sets, the kernel maintains a special internal data structure for each shared memory segment which exists within its addressing space. This structure is of type `shmid_ds`, and is defined in `linux/shm.h`.

SYSTEM CALL: shmget()

In order to create a new message queue, or access an existing queue, the `shmget()` system call is used. The first argument to `shmget()` is the key value. 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.

SYSTEM CALL: shmat()

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.

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()

This particular call is modeled directly after the *msgctl* call for message queues. 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()

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 have studied the concept of Client and Server Programming using shared memory and learned to implement Client and Server processes in the Linux kernel.