Learning Objectives: In this lab, you will learn about Interprocess communication with shared memory.

The IPC shared memory is arguably the most popular IPC mechanism. It allows processes to access some common data structures by making them available in a shared memory segment. 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.

This is often the fastest method for exchanging data between processes. Since there is no need to copy the information between the processes, the Kernel has less work to do and can operate more quickly. The most important issue raised with shared memory is synchronization The shared memory area must be managed so that a process cannot write this area when another one reads it. Semaphore can therefore be used to protect those memory areas.

Internal and User Data Structures

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 as follows:

```
/* One shmid data structure for each shared memory segment in the
system. struct shmid_ds {
        struct ipc_perm shm_perm;       /* operation perms */
        int 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 */
        unsigned short shm_cpid;        /* pid of creator */
        unsigned short shm_lpid;        /* pid of last operator */
        short shm_nattch;               /* no. of current attaches */

                                        /* the following are private */
        unsigned short shm_npages;      /* size of segment (pages) */
        unsigned long *shm_pages;       /* array of ptrs to frames ->*/
        struct vm_area_struct *attaches;        /* descriptors for attaches */
};
```

Operations on this structure are performed by a special system call, and should not be tinkered with directly.

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：

shmat() and shmdt() are used to attach and detach shared memory segments.

SYSTEM CALL: shmat();

After a shared memory region has been created with the shmget() function, the process needs to attach it to its own address space by invoking the shmat() function.

PROTOTYPE: int shmat (int shmid, char *shmaddr, int shmflg);
RETURNS: address at which segment was attached to the process, or -1 on error:

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

SYSTEM CALL: shmdt();

PROTOTYPE: int shmdt (char *shmaddr);
RETURNS: -1 on error

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.

shmctl() is used to alter the permissions and other characteristics of a shared memory segment.

SYSTEM CALL: shmctl();

PROTOTYPE: int shmctl (int shmqid, int cmd, struct shmid_ds *buf);
RETURNS: 0 on success -1 on error:

The following is a command line tool for creating, reading, writing, and deleting shared memory segments.

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#define SEGSIZE 100
main(int argc, char *argv[])
{
        key_t key;
        int shmid, cntr;
        char *segptr;
        if(argc == 1)
            usage();
       key = ftok(".", 'S');
       if((shmid = shmget(key,SEGSIZE, IPC_CREAT|IPC_EXCL|0666)) == -1)
       {
             printf("Shared memory segment exists - opening as client\n");
             if((shmid = shmget(key, SEGSIZE, 0)) == -1) {

                  perror("shmget");
                  exit(1);
             }
     }
     else
     {
                printf("Creating new shared memory segment\n");
      }
     if((segptr = shmat(shmid, 0, 0)) == -1)
     {
           perror("shmat");
           exit(1);
     }
     switch(tolower(argv[1][0]))
     {
       case 'w': writeshm(shmid, segptr,
       argv[2]); break;
       case 'r': readshm(shmid, segptr);
       break;
       case 'd': removeshm(shmid);
       break;
       case 'm': changemode(shmid, argv[2]);
       break;
       default: usage();
      }}
```

```
writeshm(int shmid, char *segptr, char  *text)
{
        strcpy(segptr, text);
        printf("Done...\n");
}

readshm(int shmid, char *segptr)
{
        printf("segptr: %s\n", segptr);
}
removeshm(int shmid)
{
        shmctl(shmid, IPC_RMID, 0);
        printf("Shared memory segment marked for deletion\n");
}
changemode(int shmid, char *mode)
{
        struct shmid_ds myshmds;
        shmctl(shmid, IPC_STAT, &myshmds);
        printf("Old permissions were: %o\n", myshmds.shm_perm.mode);
        sscanf(mode, "%o", &myshmds.shm_perm.mode); shmctl(shmid,
        IPC_SET, &myshmds);
        printf("New permissions are : %o\n", myshmds.shm_perm.mode);
}
usage()
{
        fprintf(stderr, "A utility for tinkering with shared memory\n");
        fprintf(stderr, "\nUSAGE: (w)rite <text>\n");
        fprintf(stderr, " (r)ead\n");
        fprintf(stderr, " (d)elete\n");
        fprintf(stderr, " (m)ode change <octal mode>\n");
        exit(1);
}
```

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <stdio.h>

#define SHMSZ     27

main()
{
    char c;
    int shmid;
    key_t key;
    char *shm, *s;

    key = 5678;

    if ((shmid = shmget(key, SHMSZ, IPC_CREAT | 0666)) < 0)
    {
        perror("shmget");
        exit(1);
    }


    if ((shm = shmat(shmid, NULL, 0)) == (char *) -1)
    {
        perror("shmat");
        exit(1);
    }

    s = shm;

    for (c = 'a'; c <= 'z'; c++)
        *s++ = c;
    *s  = NULL;
    while (*shm != '*')
        sleep(1);
    exit(0);
}
```

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <stdio.h>

#define SHMSZ     27

main()
{
    int shmid;
    key_t key;
    char *shm, *s;

    key = 5678;

    if ((shmid = shmget(key, SHMSZ, 0666)) < 0)
    {
        perror("shmget");
        exit(1);
    }

   if ((shm = shmat(shmid, NULL, 0)) == (char *) -1)
   {
        perror("shmat");
        exit(1);
    }

    for (s = shm; *s != NULL; s++)
        putchar(*s);
    putchar('\n');


    *shm = '*';
    exit(0);
}
```