**KING FAHD UNIVERSITY OF PETROLEUM AND MINERALS**

**Information and Computer Science Department**

# ICS 431 Operating Systems

**Lab # 10**

**Inter-Process Communication: Shared Memory**

**Objectives:**

* To learn how to create a Shared Memory Segment
* To learn how to control a Shared Memory Segment
* To learn how to attach and detach a Shared Memory Segment

**Asking for a Shared Memory Segment - shmget**:

The system call that requests a shared memory segment is shmget( ). It is defined as follows:

## shm\_id = shmget (key\_t k, int size, int flag);

In the above definition, k is of type key\_t or IPC\_PRIVATE. It is the numeric key to be assigned to the returned shared memory segment. size is the size of the requested shared memory. The purpose of flag is to specify the way that the shared memory will be used. If shmget( ) can successfully get the requested shared memory, its function value is a non-negative integer, the shared memory ID; otherwise, the function value is negative.

**Attaching a Shared Memory Segment to an Address Space- shmat**:

After a shared memory ID is returned, the next step is to attach it to the address space of a process. This is done with system call shmat( ). The use of shmat( ) is as follows:

## shm\_ptr = shmat ( int shm\_id, char \*ptr, int flag);

System call shmat( ) accepts a shared memory ID, shm\_id, and attaches the indicated shared memory to the program's address space. The returned value is a pointer of type (void \*) to the attached shared memory. Thus, casting is usually necessary. If this call is unsuccessful, the return value is -1. Normally, the second parameter is NULL. If the flag is SHM\_RDONLY, this shared memory is attached as a read-only memory; otherwise, it is readable and writable.

**Detaching and Removing a Shared Memory Segment-shmdt and shmctl:**

System call shmdt( ) is used to detach a shared memory. After a shared memory is detached, it cannot be used. However, it is still there and can be re-attached back to a process's address space, perhaps at a different address.

## shmdt (shm\_ptr);

The only argument to shmdt( ) is the shared memory address returned by shmat( ). If the detach operation fails, the returned function value is non-zero.

To remove a shared memory, use shmctl( ):

## shmctl (shm\_id, IPC\_RMID, NULL);

Where shm\_id is the shared memory ID. IPC\_RMID indicates this is a remove operation. Note that after the removal of a shared memory segment, if you want to use it again, you should use shmget( ) followed by shmat( ).

**Example#1:**

#include <sys/types.h>

#include <sys/ipc.h>

#include <sys/shm.h>

#include <sys/wait.h>

#include <stdio.h>

#include <unistd.h>

#define SHMSIZE 27

int main()

{

    int shmid;

    char \*shm;

    if(fork() == 0)

    {

        shmid = shmget(2041, SHMSIZE, 0);

        shm = shmat(shmid, 0, 0);

        printf ("Child Read <%s>\n",shm);

        shmdt(shm);

    }

    else

    {

        char c, \*s;

        shmid = shmget(2041, SHMSIZE, 0666 | IPC\_CREAT);

        shm = shmat(shmid, 0, 0);

        s = shm;

        for(c = 'a'; c <= 'z'; c++)

            \*s++ = c;

        \*s = '\0';

        printf ("Parent Wrote <%s>\n",shm) ;

        wait(NULL);

        shmdt(shm);

        shmctl(shmid, IPC\_RMID, NULL);

    }

    return 0;

}

# Example#2(Server)

#include <sys/types.h>

#include <sys/ipc.h>

#include <sys/shm.h>

#include <stdio.h>

#include <unistd.h>

#define SHMSIZE 27

int main()

{

    int shmid;

    char \*shm, \*s, c;

    shmid = shmget(5678, SHMSIZE, 0666 | IPC\_CREAT);

    shm = shmat(shmid, NULL, 0);

    s = shm;

    for(c = 'a'; c <= 'z'; c++)

        \*s++ = c;

    \*s = '\0';

    while(\*shm+25 == 'z')

        sleep(1);

    printf("From client <%s>\n",shm);

    shmdt(shm);

    shmctl(shmid, IPC\_RMID, NULL);

    printf ("Done server\n");

    return 0;

}

# Example#2 (Client)

#include <sys/types.h>

#include <sys/ipc.h>

#include <sys/shm.h>

#include <stdio.h>

#include <unistd.h>

#include <ctype.h>

#define SHMSIZE 27

int main()

{

    int shmid;

    char \*shm, \*s;

    int n;

    shmid = shmget(5678, SHMSIZE, 0666);

    shm = shmat(shmid, NULL, 0);

    printf("From server <%s>\n",shm);

    for(n = 0; n < 26; n++)

        shm[n] = toupper(shm[n]);

    shmdt(shm);

    printf ("Done client\n");

    return 0;

}

To run these programs use:

**$ gcc server.c -o server**

**$ gcc client.c -o client**

**$ ./server &**

**[1] 1104**

**$ ./client**

**From server <abcdefghijklmnopqrstuvwxyz>**

**Done client**

**$ From client <ABCDEFGHIJKLMNOPQRSTUVWXYZ>**

**Done server**

**[1]+ Done ./server**

**$**

Exercises:

Exercise 1)

Write a program in which there is a client and a server. The server requests a shared memory segment and writes 5 random integers to it. The client reads and prints these integers, then changes all of them to zero and prints their sum. The server, after finding zeros on the shared memory terminates. The client terminates too.

Exercise 2)

Write a program in which 2 processes are created: One server and one client. The server gets a shared memory segment and writes one letter into it. Then the client process tries to guess the letter by prompting the user. Each time the user makes a guess the client process informs whether the guess is too high or too low. If the user guesses correctly the client process declares him as winner terminates in less than or equal to 5 moves. If the client does not win after 5 moves, the server announces the correct letter and terminates. The client also terminates.