Shared Memory

Shared memory is the fastest form of IPC available. Once the memory is mapped into the address space of the processes that are sharing the memory region, no kernel involvement occurs in passing data between the processes. What is normally required, however, is some form of synchronization between the processes that are storing and fetching information to and from the shared memory region.

Shared memory provides a way around this by letting two or more processes share a region of memory. The processes must, of course, coordinate or synchronize the use of the shared memory among themselves. (Sharing a common piece of memory is similar to sharing a disk file, such as the sequence number file used in all the file locking examples.)

In the following example, you can see how two processes can share a common portion of the memory. Recall that when a process forks, the new child process has an identical copy of the variables of the parent process. After fork the parent and child can update their own copies of the variables in their own way, since they don't actually share the variable. Here we show how they can share memory, so that when one updates it, the other can see the change.

```
#include <stdio.h>
#include <sys/ipc.h>
#include <sys/shm.h>    /*  This file is necessary for using shared memory constructs */

main()
{
        int shmid. status;
        int *a, *b;
        int i;

        /*  The operating system keeps track of the set of shared memory  segments. In order to acquire shared memory, we must first request the shared memory from the OS using the shmget() system call. The second parameter specifies the number of bytes of
```

memory requested. shmget() returns a shared memory identifier (SHMID) which is an integer. Refer to the online man pages for details on the other two parameters of shmget() */

```
shmid = shmget(IPC_PRIVATE, 2*sizeof(int), 0777|IPC_CREAT);
                /* We request an array of two integers */
```

/* After forking, the parent and child must "attach" the shared memory to its local data segment. This is done by the shmat() system call. shmat() takes the SHMID of the shared memory segment as input parameter and returns the address at which the segment has been attached. Thus shmat() returns a char pointer. */

```
if (fork() == 0) {

        /* Child Process */
```

/* shmat() returns a char pointer which is typecast here to int and the address is stored in the int pointer b. */

```
        b = (int *) shmat(shmid, 0, 0);

        for( i=0; i< 10; i++) {
                sleep(1);
                printf("\t\t\t Child reads: %d,%d\n",b[0],b[1]);
        }
```

/* each process should "detach" itself from the shared memory after it is used */

```
        shmdt(b);

}
else {

        /* Parent Process */
```

/* shmat() returns a char pointer which is typecast here to int and the address is stored in the int pointer a. Thus the memory locations a[0] and a[1] of the parent are

the same as the memory locations b[0] and b[1] of the parent, since the memory is shared. */

```
        a = (int *) shmat(shmid, 0, 0);

        a[0] = 0; a[1] = 1;
        for( i=0; i< 10; i++) {
            sleep(1);
            a[0] = a[0] + a[1];
            a[1] = a[0] + a[1];
            printf("Parent writes: %d,%d\n",a[0],a[1]);
        }
        wait(&status);
```

/* each process should "detach" itself from the shared memory after it is used */

```
        shmdt(a);
```

/* Child has exited, so parent process should delete the cretaed shared memory. Unlike attach and detach, which is to be done for each process separately, deleting the shared memory has to be done by only one process after making sure that noone else will be using it */

```
        shmctl(shmid, IPC_RMID, 0);
    }
}
```

Questions:

1. Modify the sleep in the child process to sleep(2). What happens now?
2. Restore the sleep in the child process to sleep(1) and modify the sleep in the parent process to sleep(2). What happens now?
3. Write a C program that allows communication between a parent process and child process using Shared Memory.
   a. Parent process has to take input n from the user, where n is an integer.
   b. The parent process should then write n to the shared memory.
   c. The child process has to read the contents of the shared memory and then print all the odd numbers till the limit n.
4. Write a C program that allows communication between a parent process and child process using shared memory
   a. Parent process has to take input string from the user.
   b. The child process has to read the contents of the shared memory and convert it to capital letters and print it.

5. Write a program that creates a shared memory segment and waits until two other separate processes writes something into that shared memory segment after which it prints what is written in shared memory. For the communication between the processes to take place assume that the process 1 writes 1 in first position of shared memory and waits; process 2 writes 2 in first position of shared memory and goes on to write 'hello' and then process 3 writes 3 in first position of shared memory and goes on to write 'memory' and finally the process 1 prints what is in shared memory written by two other processes.