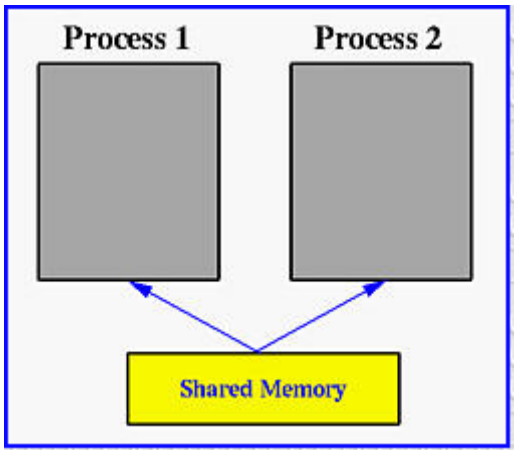
Lab 2(c):

In the discussion of the fork() system call, we mentioned that a parent and its children have separate address spaces. While this would provide a more secured way of executing parent and children processes (because they will not interfere each other), they shared nothing and have no way to communicate with each other. A shared memory is an extra piece of memory that is attached to some address spaces for their owners to use. As a result, all of these processes share the same memory segment and have access to it.



Key Points:

1. Ask for a shared memory with a memory key and memorize the returned shared memory ID. This is performed by system call shmget().
2. Attach this shared memory to the server's address space with system call shmat().
3. Initialize the shared memory, if necessary.
4. Do something and wait for all clients' completion.
5. Detach the shared memory with system call shmdt().
6. Remove the shared memory with system call shmctl().

```
#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;

        shmid = shmget(IPC_PRIVATE, 2*sizeof(int), 0777|IPC_CREAT);
                        /* We request an array of two integers */
        if (fork() == 0) {

                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 {

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

                a[0] = 0; a[1] = 1;
                for( i=0; i< 15; 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);

                shmdt(a);

                shmctl(shmid, IPC_RMID, 0);
        }
}

/*
        Try to play with this program:

        In this case we find that the child reads all the values written
        by the parent. Also the child does not print the same values
        again.

        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?

        Thus we see that when the writer is faster than the reader, then
        the reader may miss some of the values written into the shared
        memory. Similarly, when the reader is faster than the writer, then
        the reader may read the same values more than once. Perfect
        inter-process communication requires synchronization between the
        reader and the writer. You can use semaphores to do this.

        (*)Further note that "sleep" is not a synchronization construct.
        We use "sleep" to model some amount of computation which may
        exist in the process in a real world application.


*/
```