**IPC: Shared Memory**

**Subject - Unix Operating System**

**Name – Rachit Bhat**

**PRN – 22610007 Class – TYIT**

**Assignment No – 8b**

**Title-** Write 2 programs that will communicate via shared memory and semaphores. Data will be exchanged via memory and semaphores will be used to synchronize and notify each process when operations such as memory loaded and memory read have been performed.

**Objectives:**

1. To learn about IPC through message queue.
2. Use of system call and IPC mechanism to write effective application programs.

**Theory:**

Shared Memory is an efficient means of passing data between programs. One program will create a memory portion which other processes (if permitted) can access. Communication between processes using shared memory requires processes to share some variable and it completely depends on how programmer will implement it. One way of communication using shared memory can be imagined like this: Suppose process1 and process2 are executing simultaneously and they share some resources or use some information from other process, process1 generate information about certain computations or resources being used and keeps it as a record in shared memory. When process2 need to use the shared information, it will check in the record stored in shared memory and take note of the information generated by process1 and act accordingly. Processes can use shared memory for extracting information as a record from other process as well as for delivering any specific information to other process. The server maps a shared memory in its address space and also gets access to a synchronization mechanism.

The server obtains exclusive access to the memory using the synchronization mechanism and copies the file to memory. The client maps the shared memory in its address space. Waits until the server releases the exclusive access and uses the data. To use shared memory, we have to perform 2 basic steps:

* Request to the operating system a memory segment that can be shared between processes. The user can create/destroy/open this memory using a shared memory object: An object that represents memory that can be mapped concurrently into the address space of more than one process.
* Associate a part of that memory or the whole memory with the address space of the calling process. The operating system looks for a big enough memory address range in the calling process' address space and marks that address range as a special range.

**Program:**

**Server:**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <sys/ipc.h>

#include <sys/shm.h>

#include <sys/sem.h>

#include <unistd.h>

#define SHM\_SIZE 1024  // Size of shared memory

#define SEM\_KEY 1234   // Key for semaphores

#define SHM\_KEY 5678   // Key for shared memory

// Define a semaphore union

union semun {

    int val;

    struct semid\_ds \*buf;

    unsigned short \*array;

};

// Function to perform a semaphore operation (P or V)

void sem\_op(int sem\_id, int sem\_num, int op) {

    struct sembuf sop;

    sop.sem\_num = sem\_num;    // Which semaphore to operate on

    sop.sem\_op = op;          // P(sem\_num) or V(sem\_num)

    sop.sem\_flg = 0;          // No special flags

    if (semop(sem\_id, &sop, 1) == -1) {

        perror("semop failed");

        exit(1);

    }

}

int main() {

    // Create and attach shared memory segment

    int shmid = shmget(SHM\_KEY, SHM\_SIZE, 0666 | IPC\_CREAT);

    if (shmid == -1) {

        perror("shmget failed");

        exit(1);

    }

    char \*shm\_ptr = (char \*)shmat(shmid, NULL, 0);

    if (shm\_ptr == (char \*)-1) {

        perror("shmat failed");

        exit(1);

    }

    // Create semaphores for synchronization

    int sem\_id = semget(SEM\_KEY, 2, 0666 | IPC\_CREAT);  // 2 semaphores (sem1, sem2)

    if (sem\_id == -1) {

        perror("semget failed");

        exit(1);

    }

    // Initialize semaphores (sem1 = 0, sem2 = 0)

    union semun sem\_union;

    sem\_union.val = 0;

    if (semctl(sem\_id, 0, SETVAL, sem\_union) == -1 || semctl(sem\_id, 1, SETVAL, sem\_union) == -1) {

        perror("semctl failed");

        exit(1);

    }

    // Write a message to shared memory

    const char \*message = "Hello from writer process!";

    strcpy(shm\_ptr, message);  // Write to shared memory

    printf("Writer: Data written to shared memory: %s\n", message);

    // Signal sem1 to indicate that data is ready

    sem\_op(sem\_id, 0, 1);  // V(sem1)

    // Wait for reader to finish reading (sem2)

    sem\_op(sem\_id, 1, -1);  // P(sem2)

    // Detach shared memory and clean up

    shmdt(shm\_ptr);

    return 0;

}

**Client:**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <sys/ipc.h>

#include <sys/shm.h>

#include <sys/sem.h>

#include <unistd.h>

#define SHM\_SIZE 1024  // Size of shared memory

#define SEM\_KEY 1234   // Key for semaphores

#define SHM\_KEY 5678   // Key for shared memory

// Define a semaphore union

union semun {

    int val;

    struct semid\_ds \*buf;

    unsigned short \*array;

};

// Function to perform a semaphore operation (P or V)

void sem\_op(int sem\_id, int sem\_num, int op) {

    struct sembuf sop;

    sop.sem\_num = sem\_num;    // Which semaphore to operate on

    sop.sem\_op = op;          // P(sem\_num) or V(sem\_num)

    sop.sem\_flg = 0;          // No special flags

    if (semop(sem\_id, &sop, 1) == -1) {

        perror("semop failed");

        exit(1);

    }

}

int main() {

    // Create and attach shared memory segment

    int shmid = shmget(SHM\_KEY, SHM\_SIZE, 0666);

    if (shmid == -1) {

        perror("shmget failed");

        exit(1);

    }

    char \*shm\_ptr = (char \*)shmat(shmid, NULL, 0);

    if (shm\_ptr == (char \*)-1) {

        perror("shmat failed");

        exit(1);

    }

    // Create semaphores for synchronization

    int sem\_id = semget(SEM\_KEY, 2, 0666);

    if (sem\_id == -1) {

        perror("semget failed");

        exit(1);

    }

    // Wait for writer to signal (sem1)

    sem\_op(sem\_id, 0, -1);  // P(sem1)

    // Read the message from shared memory

    printf("Reader: Data read from shared memory: %s\n", shm\_ptr);

    // Signal sem2 to indicate that data has been read

    sem\_op(sem\_id, 1, 1);  // V(sem2)

    // Detach shared memory and clean up

    shmdt(shm\_ptr);

    return 0;

}

**Output:**

****

**Conclusion:**

Communication using Shared Memory IPC between client and server established and implemented using shm functions.