**VAISHNAVI E 231901059**

**DATE:19.02.2025**

**EX.NO:7**

**IPC USING SHARED MEMORY**

**Aim:**

To write a C program to do Inter Process Communication (IPC) using shared memory between sender process and receiver process.

**Algorithm:**

**sender**

1. Set the size of the shared memory segment
2. Allocate the shared memory segment using shmget
3. Attach the shared memory segment using shmat
4. Write a string to the shared memory segment using sprintf
5. Set delay using sleep
6. Detach shared memory segment using shmdt **receiver**
7. Set the size of the shared memory segment
8. Allocate the shared memory segment using shmget
9. Attach the shared memory segment using shmat 4. Print the shared memory contents sent by the sender process.

5. Detach shared memory segment using shmdt

**Program Code: sender.c**

#include <stdio.h>

#include <stdlib.h>

#include <sys/shm.h>

#include <sys/types.h>

#include <unistd.h>

#define SHM\_SIZE 1024 // Shared memory size

#define FLAG\_EMPTY 0 // Indicates shared memory is empty

#define FLAG\_FULL 1 // Indicates shared memory is full #define ARRAY\_SIZE 5 // Number of integers to send typedef struct { int status; // Flag to indicate status (empty/full) int data[ARRAY\_SIZE]; // Array of numbers

} SharedMemory; int main() {

int shmid;

SharedMemory \*shm\_ptr; key\_t key = 1234; // Shared memory key // Step 1: Create or get the shared memory segment shmid = shmget(key, sizeof(SharedMemory), 0666 | IPC\_CREAT); if (shmid == -1) {

perror("shmget failed"); exit(1);

}

// Step 2: Attach shared memory shm\_ptr = (SharedMemory \*)shmat(shmid, NULL, 0); if (shm\_ptr == (SharedMemory \*)(-1)) { perror("shmat failed"); exit(1);

}

// Step 3: Check if buffer is full if (shm\_ptr->status == FLAG\_FULL) {

printf("Error: Shared memory buffer is full. Please wait for receiver to read it.\n");

} else {

// Step 4: Get user input for numbers printf("Enter %d numbers to send: ", ARRAY\_SIZE); for (int i = 0; i < ARRAY\_SIZE; i++) { scanf("%d", &shm\_ptr->data[i]);

}

shm\_ptr->status = FLAG\_FULL; // Mark buffer as full printf("Sender: Numbers sent successfully.\n");

}

// Step 5: Detach shared memory if (shmdt(shm\_ptr) == -1) { perror("shmdt failed"); exit(1);

}

return 0;

}

**receiver.c** #include <stdio.h>

#include <stdlib.h>

#include <sys/shm.h>

#include <sys/types.h>

#include <unistd.h>

#define SHM\_SIZE 1024

#define FLAG\_EMPTY 0

#define FLAG\_FULL 1

#define ARRAY\_SIZE 5 // Number of integers to receive typedef struct {

int status; int data[ARRAY\_SIZE];

} SharedMemory; int main() {

int shmid; SharedMemory \*shm\_ptr; key\_t key = 1234;

// Step 1: Get shared memory segment shmid = shmget(key, sizeof(SharedMemory), 0666); if (shmid == -1) {

perror("shmget failed"); exit(1);

}

// Step 2: Attach shared memory shm\_ptr = (SharedMemory \*)shmat(shmid, NULL, 0); if (shm\_ptr == (SharedMemory \*)(-1)) { perror("shmat failed"); exit(1);

}

// Step 3: Check if sender has written data if (shm\_ptr->status == FLAG\_EMPTY) {

printf("Error: No numbers available. Sender has not written anything.\n");

} else {

// Step 4: Read and display numbers printf("Receiver: Received numbers - "); for (int i = 0; i < ARRAY\_SIZE; i++) { printf("%d ", shm\_ptr->data[i]);

}

printf("\n");

shm\_ptr->status = FLAG\_EMPTY; // Mark buffer as empty

}

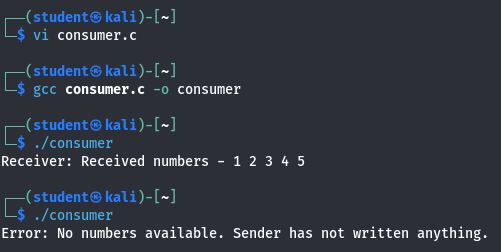
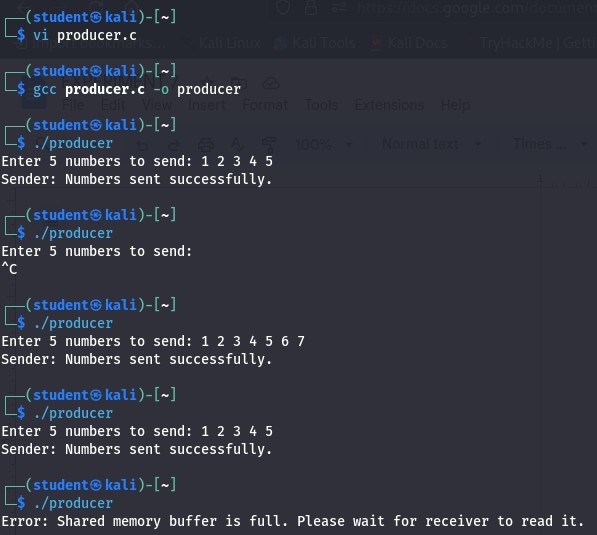
// Step 5: Detach shared memory if (shmdt(shm\_ptr) == -1) { perror("shmdt failed"); exit(1);

}

return 0;

}

**Sample Output:**



**Result:**

Hence, IPC using Shared Memory is executed successfully