**Department of Computer Science and Engineering**

**KARNATAKA LAW SOCIETY’S**

**GOGTE INSTITUTE OF TECHNOLOGY**

**UDYAMBAG,  BELAGAVI -590008**



**Course Project Report**

**Chat Application using Shared Memory**

**(16CS51)**

**Semester**: **V**

**Team Members USN**

* Nalin Patil 2GI17CS077
* Mallikarjun Kolar 2GI17CS063
* Mahantesh H 2GI17CS062
* Samarth M 2GI17CS111

**Guide:** Prof.Kuldeep Sambrekar

**Index**

**TITLE**

**PROBLEM DEFINITION**

**INTRODUCTION**

**THEROY**

**C-CODE**

**OUTPUT(SNAPSHOTS)**

**CONCLUSION**

**REFERENCES**

**Title**

**Chat Application Using Shared Memory and Signals**

**Problem Definition:**

Implementation of chat application in inter-process communication(IPC) using shared memory and signals.

**Introduction**:

Inter Process Communication through shared memory is a concept where two or more process can access the common memory. And communication is done via this shared memory where changes made by one process can be viewed by another process.

The problem with pipes, fifo and message queue – is that for two process to exchange information. The information has to go through the kernel.

* Server reads from the input file.
* The server writes this data in a message using either a pipe, fifo or message queue.
* The client reads the data from the IPC channel,again requiring the data to be copied from kernel’s IPC buffer to the client’s buffer.
* Finally the data is copied from the client’s buffer.

A total of four copies of data are required (2 read and 2 write).

Signals are a limited form of inter process communication (IPC), typically used in Unix. A signal is an asynchronous notification sent to a process or to a specific thread within the same process in order to notify it of an event that occurred. Signals originated in 1970s Bell Labs Unix and have been more recently specified in the POSIX standard.

When a signal is sent, the operating system interrupts the target process' normal flow of execution to deliver the signal. Execution can be interrupted during any non-atomic instruction. If the process has previously registered a signal handler, that routine is executed. Otherwise, the default signal handler is executed.

**Theory:**

A signal is used in the UNIX system to notify a process that a particular event has occurred. A signal may be received either synchronously or asynchronously depending on the source and the reason for the event being signalled. A signal must follow the following pattern –

1. A signal is generated by the occurrence of a particular event.

2. A generated signal is delivered to a particular process.

3. The signal must be handled after receiving at the process.

In this problem, the message is sent from one user to another user using kill function. Kill function takes two inputs – process id of the receiver process and signal type. For this purpose, we use a shared memory where we store the process id(s) of two processes. We use a handler function which will print the message received from another process. User2 will start to send message to User1 and then they will continue chatting.

So, shared memory provides a way by letting two or more processes share a memory segment. With Shared Memory the data is only copied twice – from input file into shared memory and from shared memory to the output file.

System Calls used are:

**ftok()**: is use to generate a unique key.

**shmget()**: int shmget(key\_t,size\_tsize,intshmflg); upon successful completion, shmget() returns an identifier for the shared memory segment.

**shmat()**: Before you can use a shared memory segment, you have to attach yourself  
to it using shmat(). void \*shmat(int shmid ,void \*shmaddr ,int shmflg);  
shmid is shared memory id. shmaddr specifies specific address to use but we should set  
it to zero and OS will automatically choose the address.

**shmdt()**: When you’re done with the shared memory segment, your program should  
detach itself from it using shmdt(). int shmdt(void \*shmaddr);

**shmctl()**: when you detach from shared memory,it is not destroyed. So, to destroy  
shmctl() is used. shmctl(int shmid,IPC\_RMID,NULL);

**C-Code:**

**//**User1.c

**#include <signal.h>**

**#include <stdio.h>**

**#include <stdlib.h>**

**#include <string.h>**

**#include <sys/ipc.h>**

**#include <sys/shm.h>**

**#include <sys/types.h>**

**#include <unistd.h>**

**#define FILLED 0**

**#define Ready 1**

**#define NotReady -1**

**struct memory {**

**char buff[100];**

**int status, pid1, pid2;**

**};**

**struct memory\* shmptr;**

**// handler function to print message received from user2**

**void handler(int signum)**

**{**

**// if signum is SIGUSR1, then user 1 is receiving a message from user2**

**if (signum == SIGUSR1) {**

**printf("Received User2: ");**

**puts(shmptr->buff);**

**}**

**}**

**int main()**

**{**

**// process id of user1**

**int pid = getpid();**

**int shmid;**

**// key value of shared memory**

**int key = 12345;**

**// shared memory create**

**shmid = shmget(key, sizeof(struct memory), IPC\_CREAT | 0666);**

**// attaching the shared memory**

**shmptr = (struct memory\*)shmat(shmid, NULL, 0);**

**// store the process id of user1 in shared memory**

**shmptr->pid1 = pid;**

**shmptr->status = NotReady;**

**// calling the signal function using signal type SIGUSER1**

**signal(SIGUSR1, handler);**

**while (1) {**

**while (shmptr->status != Ready)**

**continue;**

**sleep(1);**

**// taking input from user1**

**printf("User1: ");**

**fgets(shmptr->buff, 100, stdin);**

**shmptr->status = FILLED;**

**// sending the message to user2 using kill function**

**kill(shmptr->pid2, SIGUSR2);**

**}**

**shmdt((void\*)shmptr);**

**shmctl(shmid, IPC\_RMID, NULL);**

**return 0;**

**}**

**//**User2.c

**#include <signal.h>**

**#include <stdio.h>**

**#include <stdlib.h>**

**#include <string.h>**

**#include <sys/ipc.h>**

**#include <sys/shm.h>**

**#include <sys/types.h>**

**#include <unistd.h>**

**#define FILLED 0**

**#define Ready 1**

**#define NotReady -1**

**struct memory {**

**char buff[100];**

**int status, pid1, pid2;**

**};**

**struct memory\* shmptr;**

**// handler function to print message received from user1**

**void handler(int signum)**

**{**

**// if signum is SIGUSR2, then user 2 is receiving a message from user1**

**if (signum == SIGUSR2) {**

**printf("Received From User1: ");**

**puts(shmptr->buff);**

**}**

**}**

**// main function**

**int main()**

**{**

**// process id of user2**

**int pid = getpid();**

**int shmid;**

**// key value of shared memory**

**int key = 12345;**

**// shared memory create**

**shmid = shmget(key, sizeof(struct memory), IPC\_CREAT | 0666);**

**// attaching the shared memory**

**shmptr = (struct memory\*)shmat(shmid, NULL, 0);**

**// store the process id of user2 in shared memory**

**shmptr->pid2 = pid;**

**shmptr->status = NotReady;**

**// calling the signal function using signal type SIGUSR2**

**signal(SIGUSR2, handler);**

**while (1) {**

**sleep(1);**

**// taking input from user2**

**printf("User2: ");**

**fgets(shmptr->buff, 100, stdin);**

**shmptr->status = Ready;**

**// sending the message to user1 using kill function**

**kill(shmptr->pid1, SIGUSR1);**

**while (shmptr->status == Ready)**

**continue; }**

**shmdt((void\*)shmptr);**

**return 0;**

**}**

**Output :**

//Snapshots





**Conclusion:**

We understood the use of shared memory and signals for inter process communication and with the help of these concepts, we were able to do a chat application.

**References:**

<https://www.geeksforgeeks.org/ipc-shared-memory/>

<https://en.wikipedia.org/wiki/Signal_(IPC)>

<https://www.geeksforgeeks.org/chat-application-between-two-processes-using-signals-and-shared-memory/>