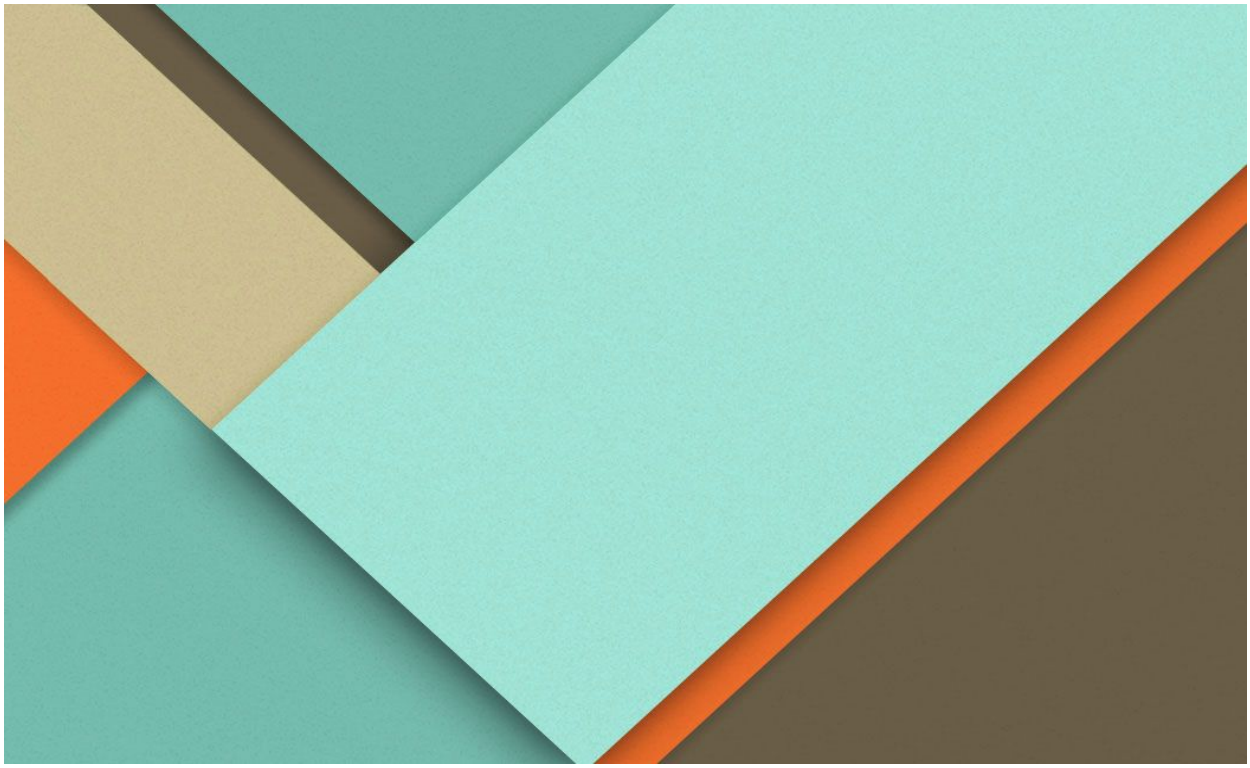


Chat Application using Interprocess Communication

04/11/2019

OPERATING SYSTEMS LAB PROJECT

Assigned TA:
Shipra Shukla

Group Members

1. Rachit Rahul Mishra (17JE003017)
2. Samyak Singh (17JE003024)
3. Vipul Bandi (17JE003026)
4. Sandesh Sinha (17JE003038)
5. Sandeep Sheela (17JE003048)
6. Avi Sahney (17JE003050)
7. Navya Srivastava (17JE003052)
8. Vipin Prakash (17JE003061)
9. Thakur Ashutosh Suman (17JE003067)

ACKNOWLEDGEMENT

We are really grateful because we managed to complete our Operating System project within the time given by our TA Shipra Shukla ma'am. This assignment cannot be completed without the effort and cooperation from our group members. We also sincerely thank ma'am for her guidance and encouragement in finishing this assignment.

We also want to thank ACS Rao sir for the valuable contribution and his guidance.

INDEX

PROBLEM STATEMENT

Project Title: Chat Application using inter-process communication

Project Description:

1. Give a brief description of the methods for inter-process communication.
2. Use one of them for designing a chat application. Multiple clients should be able to simultaneously chat with the server.
3. The user should be able to view the message history.

Inter-Process Communication

A process can be of two types namely Independent processes and Cooperating processes. An independent process is not affected by the execution of other processes while a co-operating process can be affected by other executing processes.

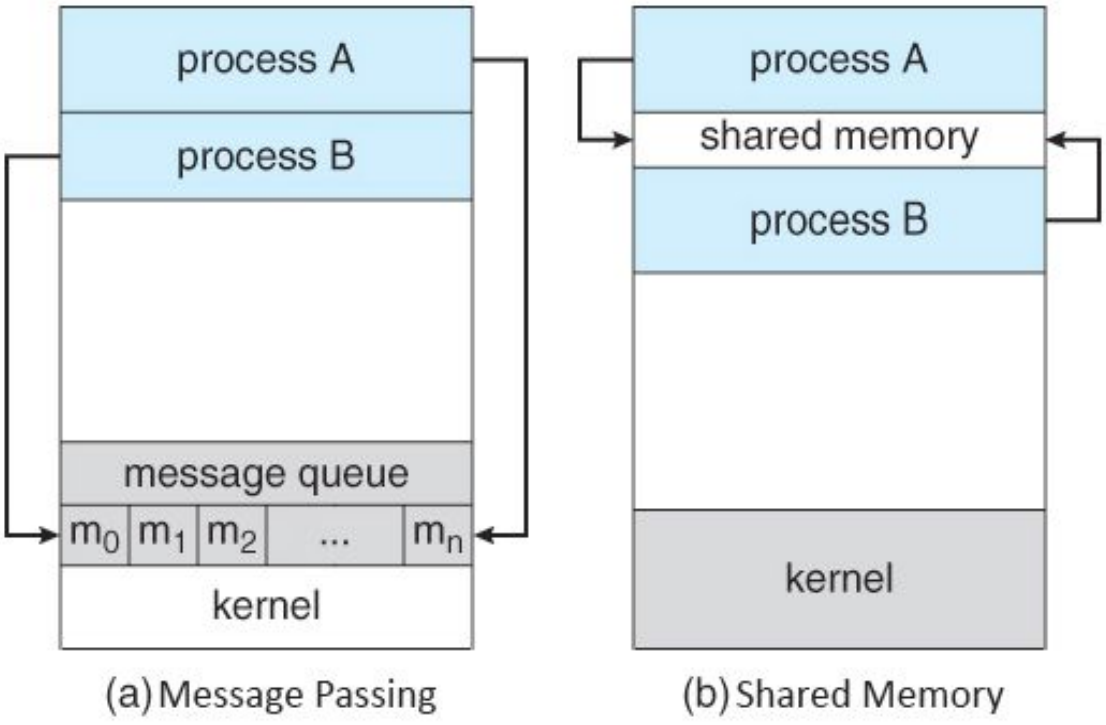
Communication can be of two types −

- Between related processes initiating from only one process, such as parent and child processes.
- Between unrelated processes, or two or more different processes.

Interprocess communication (IPC) is a set of programming interfaces that allow a programmer to coordinate activities among different program processes that can run concurrently in an operating system. This allows a program to handle many user requests at the same time. Since even a single user request may result in multiple processes running in the operating system on the user's behalf, the processes need to communicate with each other. The IPC interfaces make this possible. Each IPC method has its own advantages and limitations so it is not unusual for a single program to use all of the IPC methods.

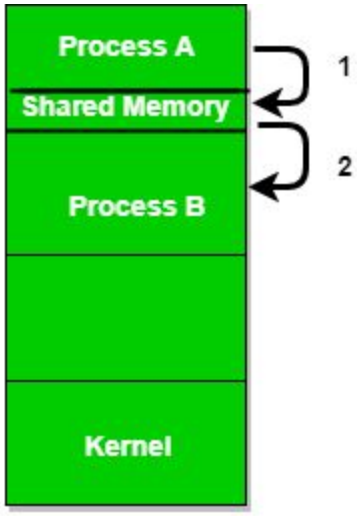
Inter-process communication (IPC) is a mechanism which allows processes to communicate with each other and synchronize their actions. The communication between these processes can be seen as a method of co-operation between them. Processes can communicate with each other using these two ways:

1. Shared Memory
2. Message Passing



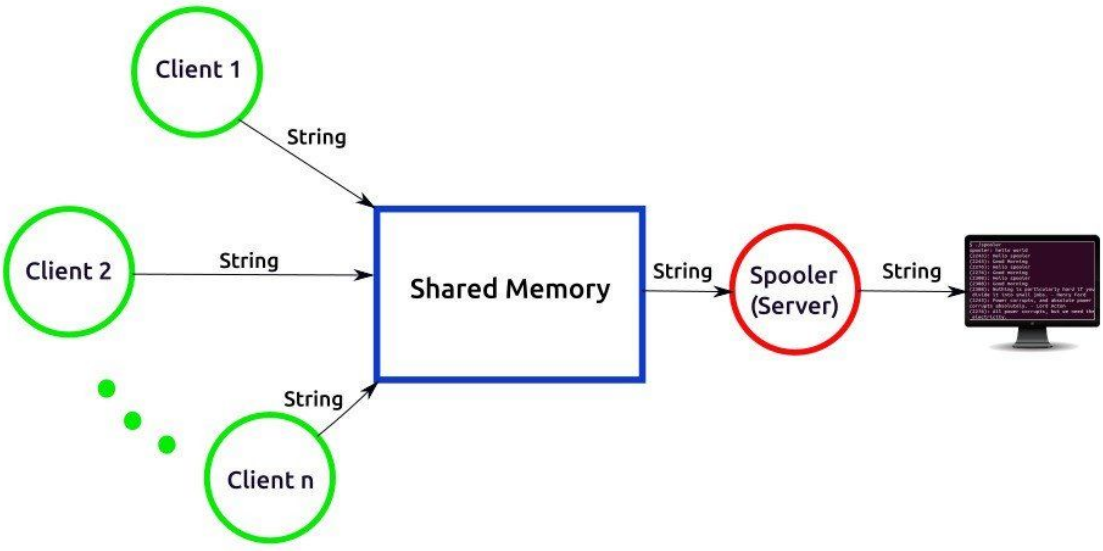
Shared Memory Concept

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. Once created, a **shared** segment can be attached to a process address space using shmat(). It can be detached using shmdt().



OUR APPROACH

The problem statement requires the server to handle multiple clients messages and communicate with them. We have designed a system where messages of multiple clients are displayed at the server's end and the server can respond to them. In this system, the server reads the message once a client writes and then waits for another client to write. If no client writes to the shared memory location in a given amount of time the server again takes the control. The message written by the server is displayed to all the clients.



Software Architecture for Clients and Server using Shared Memory

Sending and receiving messages:

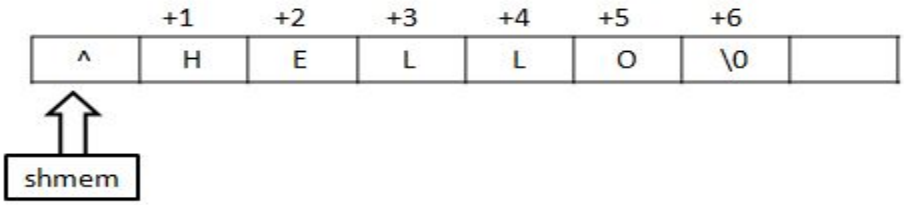
We have assigned a common port number to all the clients and need not be given during input but to differentiate between different clients, we need to enter a different number corresponding to ./a.out as an argument to char *argv[]. This can be done as shown below.

```
>> g++ client.cpp
>> ./a.out 1
```

This helps the server to identify each client differently and receive their messages. First, we run the server and create a shared memory ID using shmget((key_t)1080, 27, IPC_CREAT | 0666), which is further used by shmat() to attach the shared memory to the program. Once this shared memory segment is generated different clients can attach the shared memory using the same shared memory ID (they need not create the shared memory ID again)

The communication is started by the server by writing the message at the shared memory location. (for example, let the message length be l and the address of shared memory location be stored by a char* shmem. Then the message is written at shmem+1 to shmem+l). Once

the message is written by the server, the values stored at shmem is changed to ^, which indicates that the server has written its message. Then the control is given to the clients to read the message and write the response.



Each client is given 5 seconds to respond to the message. Once a client writes the message the value at shmem is changed to the client number (given as an argument to char *argv[]). This allows the server to identify which client has written the message. If no message is written in this 5 seconds by any client the control again returns to the server. If any of the clients or server wants to close the connection, they can enter * instead of the message.

How does the server know that all the clients have responded or no clients have anything further to respond to the message?

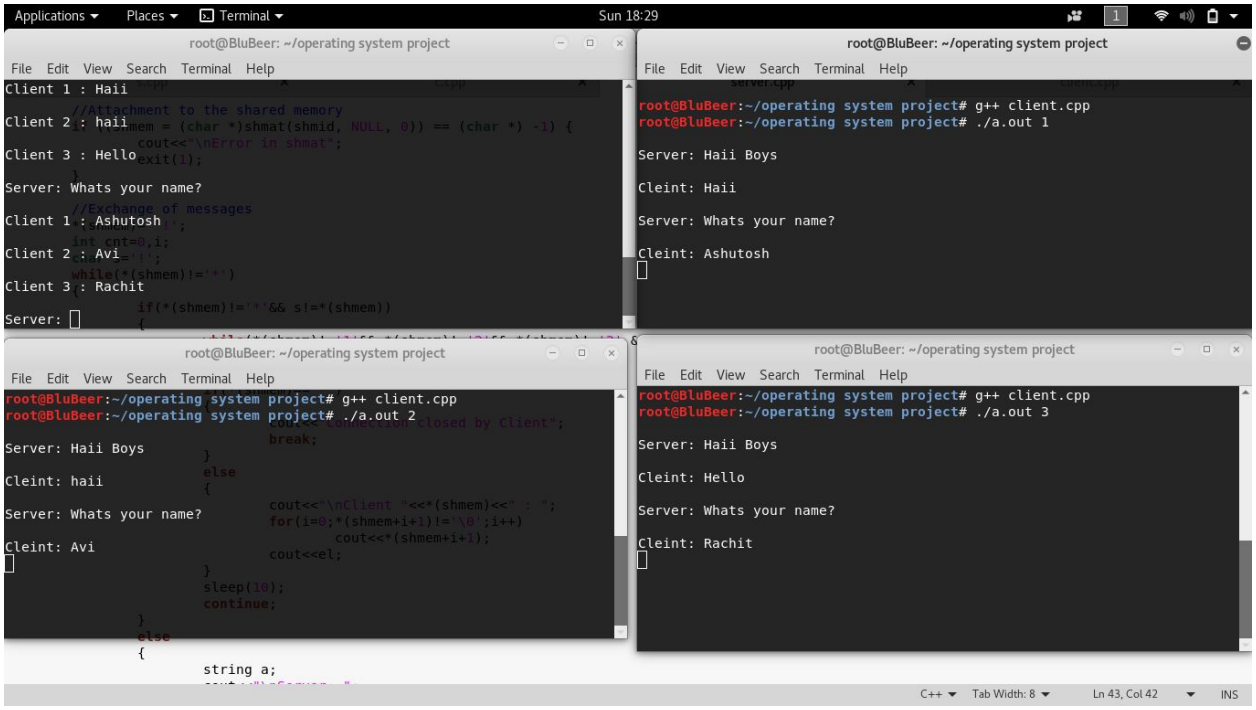
When the server read the message of the clients it stores the number of the client who has written that message and after reading the message it waits for 5 seconds for any further response and if the value at shmem still remains the same then the server takes the control from the clients.

Reading the message history:

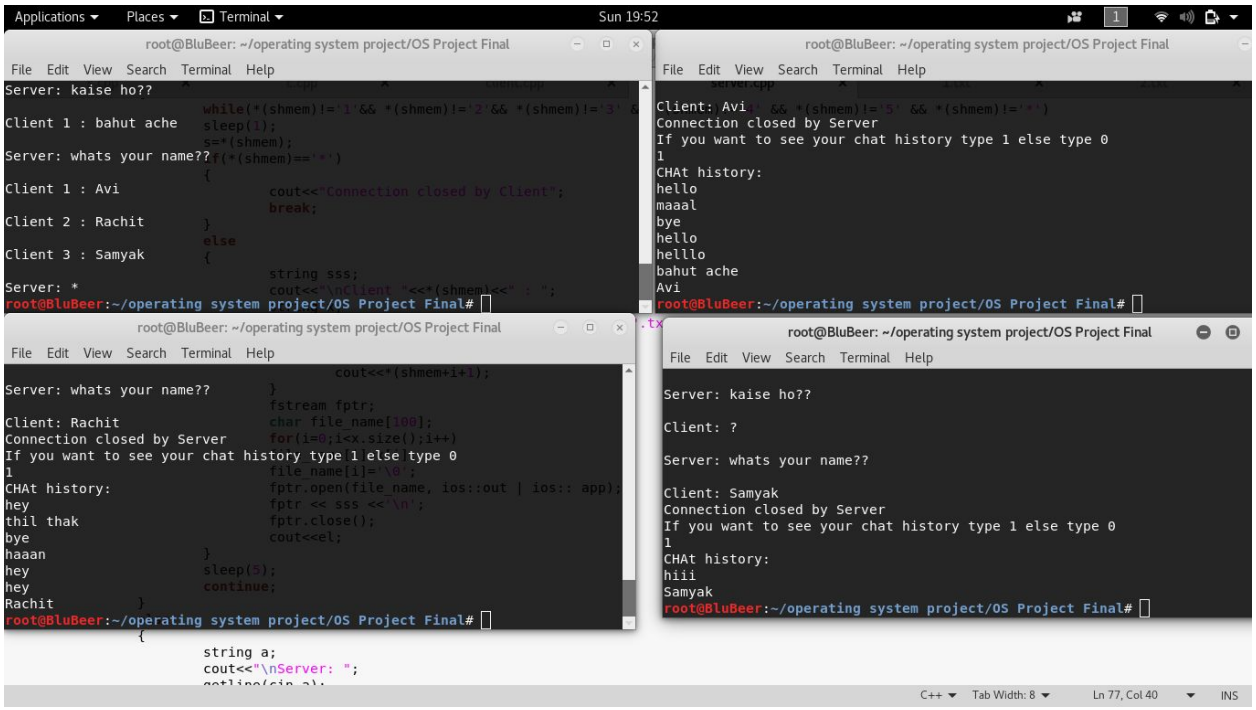
When the server reads the messages of different clients, it also stores these messages to a text file which clients can access to see their message history. Each client has its own message history text file and can access only that particular file. The message history can only be displayed after the chatting connection has been closed.

Working Screenshots of Output

1. 3-clients in a chat with the server



2. Chat History



Limitations

1. There a strict allocation of time in which a client needs to respond to the message.

2. Only one client can send a message to the server in the given span of 5 seconds (span can be adjusted). If another client tries to send the message in that span of time the message is overwritten.