Assignment No : 01

1) Title : Implementation of Inter-Process-Communication using socket programming : implementing multithreaded echo server.

2) Software / Hardware Requirement :

* Software Requirement
  1) Unix / Linux Distro : Ubuntu
  2) Code Editor

* Hardware requirement
  1) Computer System
     Process : 15th gen
     Ram : 8 GB.
  2) I/O Peripherals : keyboard / Mouse
  3) Monitor : 720p / 1080p FHD / IPS.

3) Learning Objective :
   1) To understand inter-process communication and sockets.
   2) To apply engineering background and skills to solve given problem.
   3) To use appropriate datastructure.

4) Learning Outcome :
   1) Understood the working of ~~proc~~ IPC (inter-process communication)
   2) To understand synchronous & asynchronous communication

3) Applied appropriate data structure to solve given problem.

5) Concept Related Theory:

What is IPC?

In computer science, inter-process communication or interprocess communication (IPC) refers specially to the mechanisms an operating system provides to allow the processes to manage shared data. Typically, application can use IPC, categorized as clients and servers, where the client request data and the server responds to client request. Many applications are both clients and server, as commonly seen in distributed computing.

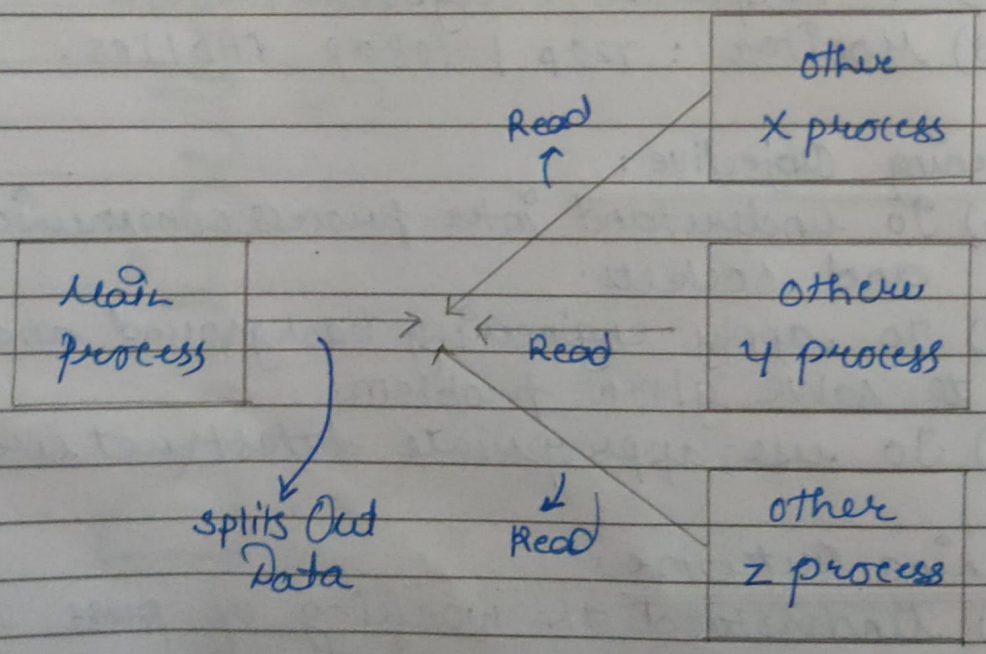

fig. IPC

IPC is very important to the design process for microkernels, and nano kernels, which reduces the number of functionalities provided by the kernel. Those functionalities are then obtained by communicating with servers via IPC, leading to a large increase in communication when compared to a regular monolithic kernel. IPC interfaces generally encompass variable analytic framework structure. These process ensure compatibility between the multi-vector protocols upon which IPC models rely.

An IPC mechanisms is either synchronous or asynchronous. Synchronous. Synchronization primitives may be used to have synchronous behaviour with an asynchronous IPC mechanism.

Synchronous communication:

It happens when messages can only be exchanged in real time. It requires that the transmitter and receiver are present in the same time and/or space. Examples of synchronous communication are phone calls or video meeting

Asynchronous communication

It happens when information can be exchanged independent of time. It doesn't require the recipients immediate attention allowing them to respond to the message at their convenience. example of asynchronous communication are emails, online forums, and collaborative documents.

What is a Socket?

A socket is one endpoint of a two-way communication link between two programs running on the network. A socket is bound to a port number so that the TCP layer can identify the application that data is designed to sent to.

What is a port?

A port is a virtual point where network communication starts and end. Ports are software based and managed by a computer's operating system. Each port is associated with specific process or service.

6) Algorithms

*) Server

a) Start
b) initialize ip and port for server.
c) create socket variables.
d) Start TCP server socket
e) Listen for client request
f) Print Message received from client
g) Echo the message back to client.
h) Close socket connection
i) End.

*) Client

a) Start
b) initialize ip and port for client side
c) create socket variables.
d) Start TCP socket.
e) Connect to the servers given socket and port.
f) Accept the data from user and send it to the server.
g) Close the connection
h) End.

7) Conclusion:

Understood the working of inter process communication, asynchronous communication as well as synchronous communication.

8) References:

1) Geeks for Geeks
2) Youtube.

Code for server (Server.c)

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <arpa/inet.h>

#define PORT 5566

int main(){

    int sockfd, ret;
    struct sockaddr_in serverAddr;

    int newSocket;
    struct sockaddr_in newAddr;

    socklen_t addr_size;

    char buffer[1024];
    pid_t childpid;

    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    if(sockfd < 0){
        printf("[-]Error in connection.\n");
        exit(1);
    }
    printf("[+]Server Socket is created.\n");

    memset(&serverAddr, '\0', sizeof(serverAddr));
    serverAddr.sin_family = AF_INET;
    serverAddr.sin_port = htons(PORT);
    serverAddr.sin_addr.s_addr = inet_addr("127.0.0.1");

    ret = bind(sockfd, (struct sockaddr*)&serverAddr, sizeof(serverAddr));
    if(ret < 0){
        printf("[-]Error in binding.\n");
        exit(1);
    }
    printf("[+]Bind to port %d\n", PORT);
```

```
    if(listen(sockfd, 10) == 0){
        printf("[+]Listening....\n");
    }else{
        printf("[-]Error in binding.\n");
    }


    while(1){
        newSocket = accept(sockfd, (struct sockaddr*)&newAddr, &addr_size);
        if(newSocket < 0){
            exit(1);
        }
        printf("Connection accepted from %s:%d\n", inet_ntoa(newAddr.sin_addr),
ntohs(newAddr.sin_port));

        if((childpid = fork()) == 0){
            close(sockfd);

            while(1){
                recv(newSocket, buffer, 1024, 0);
                if(strcmp(buffer, ":exit") == 0){
                    printf("Disconnected from %s:%d\n", inet_ntoa(newAddr.sin_addr),
ntohs(newAddr.sin_port));
                    break;
                }else{
                    printf("Client: %s\n", buffer);
                    send(newSocket, buffer, strlen(buffer), 0);
                    bzero(buffer, sizeof(buffer));
                }
            }
        }

    }

    close(newSocket);


    return 0;
}
```

Code for Client (Client.c)

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <arpa/inet.h>

#define PORT 5566

int main(){

    int clientSocket, ret;
    struct sockaddr_in serverAddr;
    char buffer[1024];

    clientSocket = socket(AF_INET, SOCK_STREAM, 0);
    if(clientSocket < 0){
        printf("[-]Error in connection.\n");
        exit(1);
    }
    printf("[+]Client Socket is created.\n");

    memset(&serverAddr, '\0', sizeof(serverAddr));
    serverAddr.sin_family = AF_INET;
    serverAddr.sin_port = htons(PORT);
    serverAddr.sin_addr.s_addr = inet_addr("127.0.0.1");

    ret = connect(clientSocket, (struct sockaddr*)&serverAddr,
sizeof(serverAddr));
    if(ret < 0){
        printf("[-]Error in connection.\n");
        exit(1);
    }
    printf("[+]Connected to Server.\n");

    while(1){
        printf("Client: ");
        scanf("%s", &buffer[0]);
        send(clientSocket, buffer, strlen(buffer), 0);
```

```
        if(strcmp(buffer, ":exit") == 0){
            close(clientSocket);
            printf("[-]Disconnected from server.\n");
            exit(1);
        }

        if(recv(clientSocket, buffer, 1024, 0) < 0){
            printf("[-]Error in receiving data.\n");
        }else{
            printf("Server: %s\n", buffer);
        }
    }

    return 0;
}
```

Output: