UNIT 5

=> Define parent and child process with eg (3)

=> Differentiate between SOCK_STREAM () & SOCK_DGRM() (3)

=> Define the Concept of elementary Socket-System call (10)

=> Explain in detail the IPC pgmming with eg (10)

=> Define Inter process Communication with eg (5)

=> htons() ntons() System call (3)

=> Define Socket. Discuss the Connected oriented and Connectionless Socket System calls briefly (10) (10)

=> Client Server pgm with Suitable eg. (10)

Interprocess Communication (IPC)

⇒ Interprocess Communication is the mechanism which allows processes to communicate with each other and to synchronise their actions.

⇒ ~~IPC is an important so he causes.~~

~~Information Sharing~~ why IPC?

⇒ IPC is an important Concept.

Suppose process may be present with in one system or within another system with the help of networking they will be communicated. In IPC Information Sharing will be there and resource sharing. It means one system or within one system. one can share a resource also with in another system.

⇒ with the help of IPC Computation Speed will be increased, if one use limited resources, if this resource will be sharing by other process also, hence IPC plays important role here

⇒ it having the modularity & convience

⇒

⇒ The IPC this allows a program to handle many user request at same time.

Basically the IPC is process to process communication within or with in an operating system. The process is nothing but it is a series of action or steps in order to achieve a predicted end.

Process can be two type:

* Independent process
* Cooperative process

An independent process, it means it is not affected by the execution of other processes, while Cooperative, it can be process can be affected by other executing program.

⇒ In Interprocess Communication, the process can be communicated with each other by using two ways.

* Shared memory
* Message passing

1) Shared memory

Shared memory is memory shared between two or more cpp process

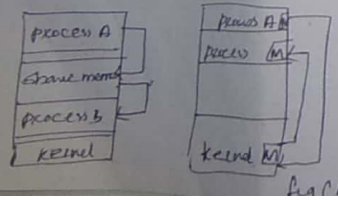


fig (A) fig (B)

In fig (A) here process A generate information about certain resources and keep it as a record in shared memory. In that record the resources that is used by the process A will be saved in the shared memory. Suppose the process B wanted to use that shared memory, by firstly it check the record for identify what type of data are sharing by the process A. So, process B check the information and make a note of information generated by process A. If process A take a resources instead of that resources process B take another resources, so this is the cooperation between the two process.

In fig b each process, having the shared memory, whenever the process A use some shared memory it sends the information to the OS, then the OS storing the process A information of process A. Whenever the process B wanted to do some operation it first check the OS, if any other device is using the resources or not.

If it is wrong, ...
another resources.

Message passing

In message passing there is no use of shared memory. The process communicated with each other without using any kind of shared memory.

eg: Suppose two process are there P1 & P2 wants to be communicate

1) First establish a communication link
2) Then exchange msg using basic primitives
   Primitives are send and receive.

The msg passing may be classified direct or indirect communication

IPC Methods

Pipe: Communicate between two related process. This mechanism is half duplex, meaning - The first process communicates with second process. To achieve a full duplex i.e - the second process communicate with first process another pipe is required

FIFO: Communicate between two unrelated process. FIFO is full duplex, meaning first process can communicate with the second process and vice versa

Message passing

Message Queue: Communication between two or more process with full duplex - capacity. The process will communicate with each other by posting a message and retrieving it out of the queue. Once - retrieved, the message is not longer available in the queue.

Shared memory: Communicate between - two or more process is achieved through - shared piece of memory among all process.

Semaphores: More than process can communicate the semaphore the process when one process wanted process the - memory (for reading or writing), it needs to be locked (or protected) and released when the access is removed

Signal: Communicate multiple process by the way of signaling. This means a source process will send a signal and destination process will handle it.

Process Creation & Termination

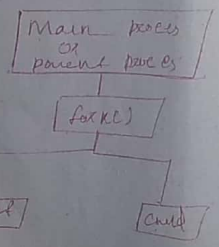
Process is a series of action / step — in order to achieve particular end.

Whenever we execute a program — then a process is created and would be terminated after the completion of the execution. Each process is identified with a unique positive integer — called as process ID or simple PID

The process creation is achieved through the fork() system call.

The newely created process is the child process and the process which — execution is stated is called parent process

After the system call we have two — process parent and child



- The fork() system call returns three values

1) Negative value to indicate an error, i.e, unsuccessful when creating the child process.

2) Returns a zero for child process.

3) Returns a positive value for the parent process.

eg:
```
#include<stdio.h>
#include<sys/types.h>
#include<unistd.h>
int main()
{
    fork();
    printf("Hello world\n");
    return 0;
}
```

o/p

```
Hello world
Hello world
```

```
#include (stdio.h)
#include <sys/types.h>
int main()
{
    fork();
    fork();
    fork();
    printf("hello \n");
    return 0;
}
```

hello
hello
hello
hello
hello
hello
hello
hello

Total no. of procs = 2^n

where n is the fork() system call

So,

$n = 3 \Rightarrow 2^3 = 8$

PIPE

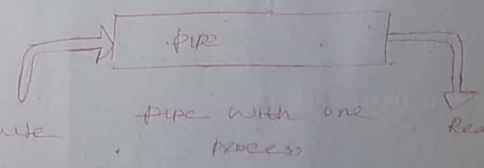
⇒ Pipe one way communication only

⇒ Pipe communicate two related process. It is half duplex. It achieved that means The first process is communicated with the second process, it achieved full duplex, that means second process communicated with the first process by using another pipe.

⇒ Pipe is one way communication only. We can use pipe such that one process to write to pipe.

⇒ pipe is one way communication only

⇒ To achieve pipe system call, create two file one for write into the another is read from the file



There is two descriptors

First one is connected to the read point - pipe and another is connected to write end to pipe.

Descriptor pipedes[0] is for reading

Descriptor pipedes[1] is for writing.

=> The pope() create a two pipe

=> The pclose() closed a pipe

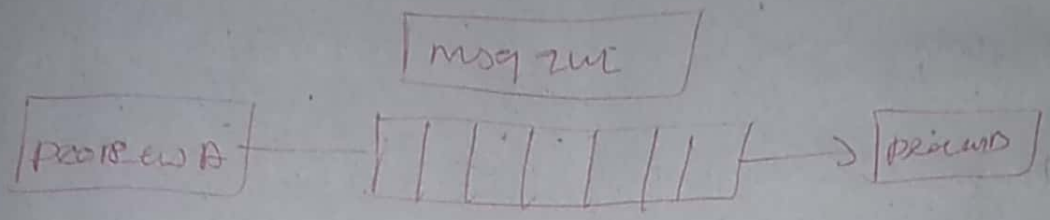
Message Queue

Communication between two process - between full duplex capability. The - full duplex means first process is - communicated with second process vice versa. The process will communicate with each - other while posting a message and - retrieved it out of queue. once the - msg is retrieved it has no longer available in the queue.

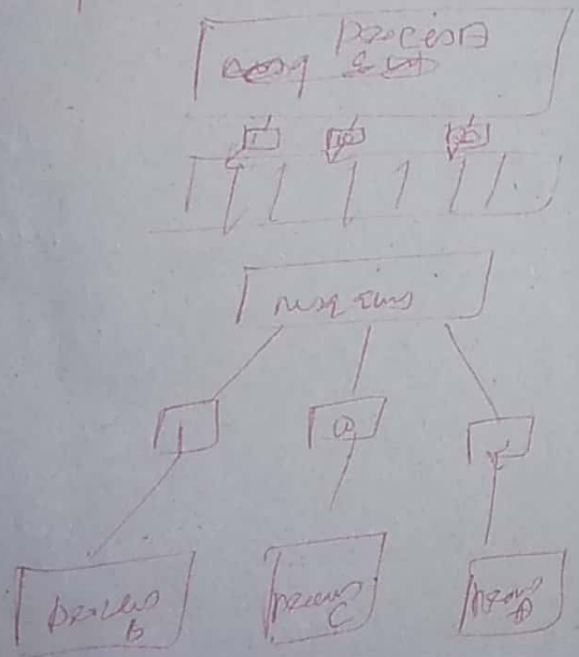
=> There is no synchronization.

Message Queue

writing into the shared memory one process and reading from the shared memory — another process



writing is into shared memory by one process with current data packet and reading from by multiple process



data packets are at a time only one process can read.

To perform communication using message queue

1) Create a message queue or connect to an already existing msg (msgget())
2) write into the msg queue (msgsnd())
3) read from the msg queue (msgrcv())
4) perform control operations on the msg queue (msgctl())

Semaphore

There are two types

Binary Semaphore: It is special form of semaphore used for implementing mutual exclusion, hence it is often called mutex. A binary semaphore is initialized to 1 and only takes the value 0 during execution of prgm

Counting Semaphore

These are used to implement bounded concurrency

Socket

⇒ Socket allow communication between two different process on the same or different machine.

⇒ The communication done with the help of by using standard unix file descriptor.

⇒ In unix, every input/output action done by writing or reading a file descriptor.

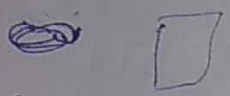
⇒ The socket is a low level file descriptor

⇒ This is because command such as read() & write() work with socket in the same way they do with files and pipes.

⇒ Socket is a end point of communication between 2 device.

eg: when we have two persons at different places and they want to communicate with each other. Each of them have mobile phones, one of them will have to initiate the communi-

cation by dialing the other persons number and after the other person accept the call then Connection can be established. Once the Connection is established they can start talking to each other by sending and receiving msg's, Now in this case each of these mobile phone acts like a Socket, which simply an end point of Communication



Where Socket Used?

⇒ Unix Socket is used in client-server application frame. A server is process, client request is processed by server

⇒ Most of the application level protocols like FTP, Pop3, SMTP these are used in Socket, to establish Connection between client and server and for exchanging data.

Types of Socket

There are four types of Sockets available to the users. The first two are most commonly used and the last two rarely used.

1) Stream Socket :-

The Stream Sockets are provided ~~guaa~~ guaranteed ~~deliv~~ delivery. In Stream Socket, that ever the ~~data~~ order you send data through the stream socket then it reach at the destination at the same order ~~that you sent~~

ie, If Stream Socket through send three item "A, B, C" then they reach at the destination in the same order "A, B, C"

=> The Socket TCP for data transmission

=> If delivery is impossible, the Sender receives an error indication

~~Datagram Socket~~

Connectionless

Datagram Socket:

In datagram socket delivery of is not guaranteed. It is connectionless there is no any connection between the socket. They use UDT (User datagram packet).

Raw Socket

These provide user access to the underlying communication protocol, which support socket abstraction. These socket are normally datagram oriented.

Sequenced packet Socket

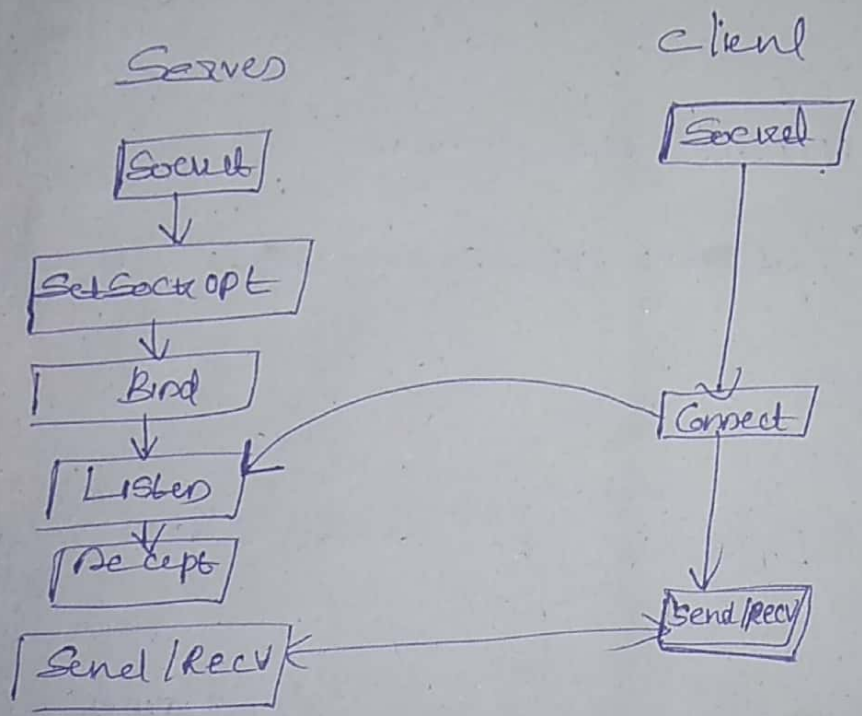
This similar to stream socket, here in sequence packet data boundaries are preserved, in the case of stream socket the data receives do not have any bound that is only the amount is sent.

Socket

Socket programming

⇒ Socket programming is a way of connecting two nodes on network to communicate with each other

⇒ one socket(node) listen to a particular port of a IP. while other socket reaches out to get a connection. Server listen to all client but client reaches out to server



Stages for Server

Socket Creation:

int sockfd = Socket(domain, type, protocol)

sockfd: Socket descriptor

domain: Communication domain

Eg: AF_INET (IPv4 protocol)
AF_INET6 (IPv6 protocol)

type: Communicate type

SOCK_STREAM : TCP (reliable, Connection oriented)

SOCK_DGRAM : UDP (unreliable, Connectless)

protocol : value for IP

Setsockopt

socket option are defined

int Setsockopt (int sockfd, int level, int optname,

Bind

Bind function bind to the Server Client Socket

int bind (int sockfd, const struct sockaddr *addr, socklen_t addrlen)

Listen

Listen the Servs and client

int listen(sockfd, int backlog)

Accept

To accept the Connection, The new Connection can accept

Stage for Client

Socket Connect : Concretely Similar to Servers Socket Creation

Connect (int

Connect

int Connect(int sockfd, const struct sockaddr *addr, socklen_t addrlen);

here First Create the Socket and the Set the option then bind to the client socket, the listen then listen to the client and serves. Then client Conn Connected to listen, Also Listen Servs accept and the send/receive

Elimentary System Call for Socket

Socket
The Create an end point between Client & Server.

Bind
Binding assigning an ip address

Listen
Waiting for Connection
=) Listen basically done Server only.
Connection oriented Servie ready to Recieve.
(End Socket, End back log)

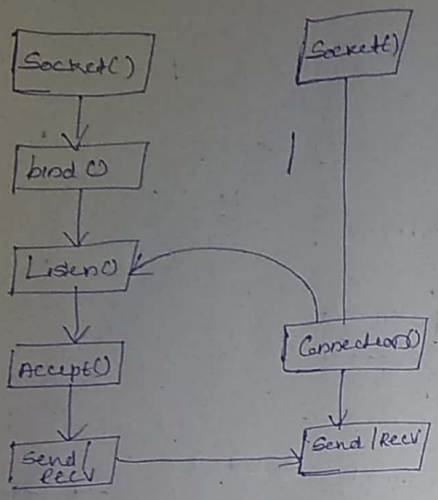
Accept
It waiting for someno to Connect
=) Accept new Connection
int accept (int Socket, (Struct Sockaddr *
addr, Socklen-t- addrlen);

Send / Recv
Send / Recv
Sending data Recving data

Tcp chat program
Transmission Control protocol
Connet oriented

Server
1) Create a Socket with Socket()
2) Bind the Socket to an address bind()
3) Listen the Connection with the listen()
3) Accept the Connection with Accept()
4) Send and Recieve data use Read() and Write() System Call

client
1) Create the Socket with Socket()
2) Connect the Socket to the address of the Server using client() System call.
3) Send and recieve data using Read()
Sept write() System



The ~~Con~~ TCP mean TRANSMISSION-Control protocol, then it is Connection-oriented. In Connection oriented, Initially the Connection is established. ~~First the server is ready~~ First the server is ready, the after some time client is started.

UDP Chat

⇒ User Datagram protocol

Connection ~~oriented~~ less service

In UDP chat client does not create a Connection between the server. It just send data directly. There is no any acknowledgment given

Server

1) Create ~~Socket~~ with the Socket()
2) Bind the Socket to an addrs using the bind()
3) Send and recieve data use the recvfrom() and Sendto() System call.

Client

⇒ Create a Socket with Socket()
⇒ Send and receive data use recvfrom()- and Sendto()

System Calls

Sock Creation

~~Socket~~

~~Socket = int sockfd~~

int sockfd = Socket(int domain, int type, int protocol);

~~int pro~~

domain

Communication domain

eg: AF_INET (IPV4)
AF_INET6 (IPV6)

Type => Communication type

Socket_STREAM()

Socket_DGRAM

Protocol

value of IP

0 means default protocol.

Bind

int bind(int sockfd, ~~struct~~ Char ~~struct addr~~ *addr, ~~sock~~ socklen_t addrlen)

Sockfd: File descriptor socket to be bound

addr: address

addrlen: size of address structure

~~Listen~~ Recev

~~recv from()~~

int recvfrom(int sockfd, void *buf, size_t len, int flags, struct sockaddr *src_addr, socklen_t *addrlen)

buf: Application buffer in which to receive data.

len: size of buf

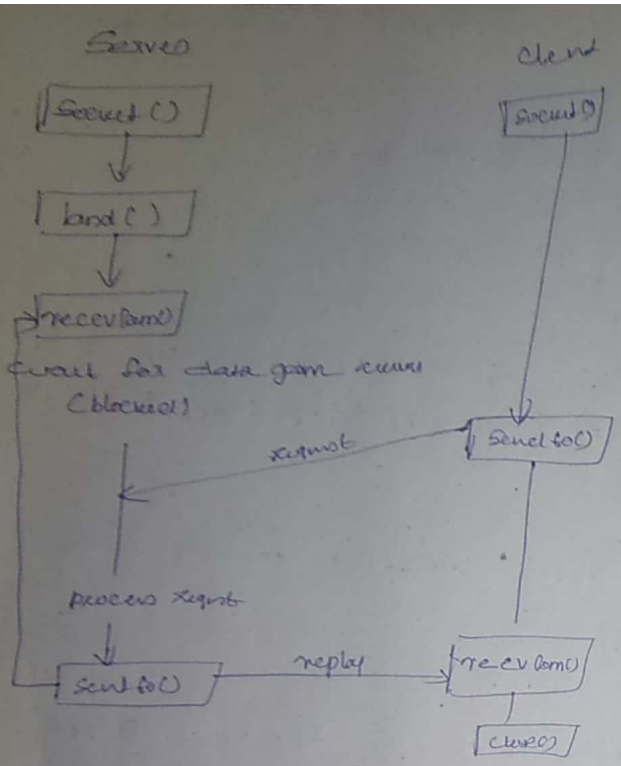
Flags: Bitwise OR of flags to modify socket behaviour

src_addr: structure containing source address is returned

addrlen: variable in which size.

Sendto()

int sendto(int sockfd, const void *buf, size_t len, int flags, const struct sockaddr *dest_addr, socklen_t addrlen)



Initially the Connection less it will be create it own socket, then end point is created then address is assigned to end point by using bind() system call. Then it receive ready to receive from anywhere. Assume that client is ready and he binded to the IP addrs then he send directly to request to the server side. Then the server will process the request then it basically sent to the client then it received by the client, then the thus process continues until the requirement. once the everything is done they will close both the Connection using close().

eg:

```
#include <stdio.h>
#include <string.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netnet/in.h>
#include <arpa/inet.h>

main()
{
  Struct SockaddrIN client, server;
  int s, n;
  Char b1[100], b2[100];
  S=Socket(AF_INE, SOCK_DGRAM, 0)
  Server.SIN_family = AF_INET
  Server.SIN_port = 2000
```