IPC Techniques – Sockets

- Unix/Linux like OS provide Socket Interface to carry out communication between various types of entities
- The Socket Interface are a bunch of Socket programming related APIs
- We shall be using these APIs to implement the Sockets of Various types
- In this course We shall learn how to implement Two types of :
  - Unix Domain Sockets
    - IPC between processes running on the same System
  - Network Sockets
    - Communication between processes running on different physical machines over the network
- Let us First cover how Sockets Works in General, and then we will see how socket APIs can be used to implement a specific type of Communication. Let us first Build some background . . .

IPC Techniques – Sockets

1.2

- Socket programming Steps and related APIs
- Steps :

1. Remove the socket, if already exists
2. Create a Unix socket using *socket()*
3. Specify the socket name
4. Bind the socket using *bind()*
5. listen()
6. accept()
7. Read the data recvd on socket using recvfrom()
8. Send back the result using *sendto()*
9. *close* the data socket
10. *close* the connection socket
11. Remove the socket
12. exit

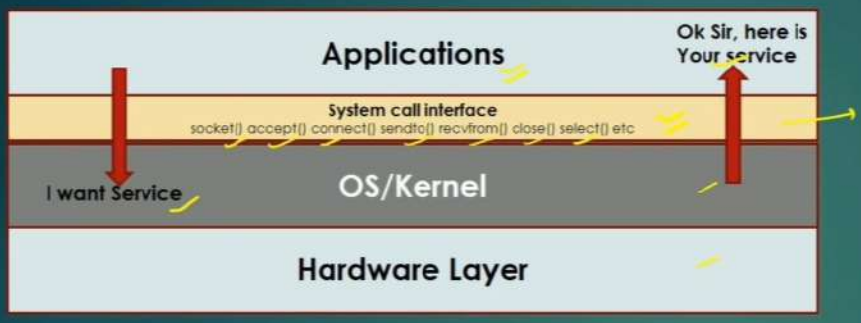
Before diving into these steps, let's study *how the Socket based communication state machine works*

and

various socket APIs provided by Linux OS

Linux System call Interface

1.3



Computer Layer Architecture

- Linux Provides a set of APIs called System calls which application can invoke to interact with the underlying OS
- Socket APIs are the interface between application and OS
- Using these APIs, application instructs the OS to provide its services
- Familiar Example :
  - malloc(), free()

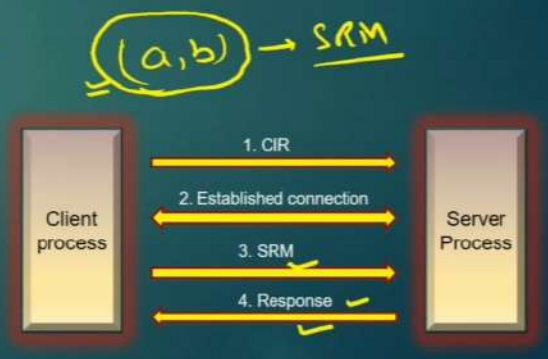
Socket Message types

- Messages (Or requests) exchanged between the client and the server processes can be categorized into two types :
  - Connection initiation request messages
    - This msg is used by the client process to request the server process to establish a *dedicated* connection.
    - Only after the connection has been established, then only client can send Service request messages to server.

And

  - Service Request Messages
    - Client can send these msg to server once the connection is fully established.
    - Through these messages, Client requests server to provide a service

- Servers identifies and process both the type of messages very differently



State machine of Socket based Client Server Communication

1. When Server boots up, it creates a *connection socket* (also called *"master socket file descriptor"* using *socket()*)

   M = *socket()*

2. M is the mother of all Client Handles. M gives birth to all Client handles. Client handles are also called *"data sockets"*

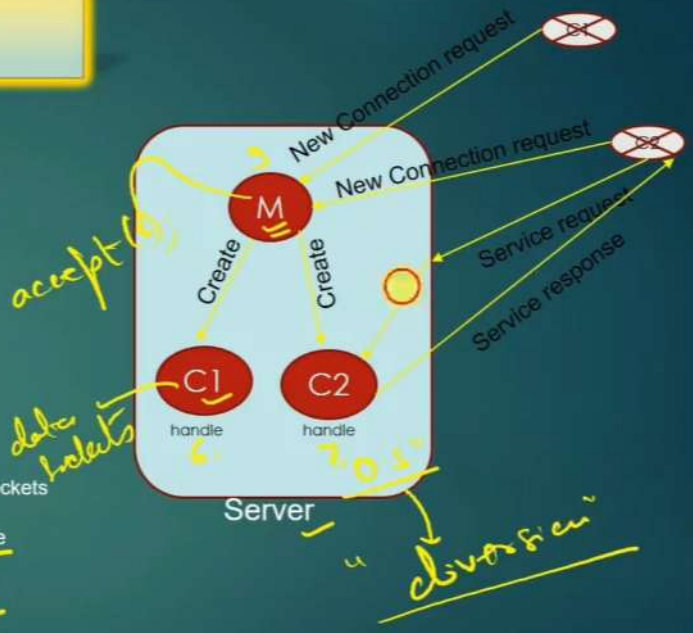
3. Once Client handles are created for each client, Server carries out Communication *(actual date exchange)* with the client using Client handle (and not M).

4. Server Has to maintain the database of connected client handles Or data sockets

5. M is only used to create new client handles. M is not used for data exchange with already connected clients.

6. *accept()* is the system call used on server side to create client handles.

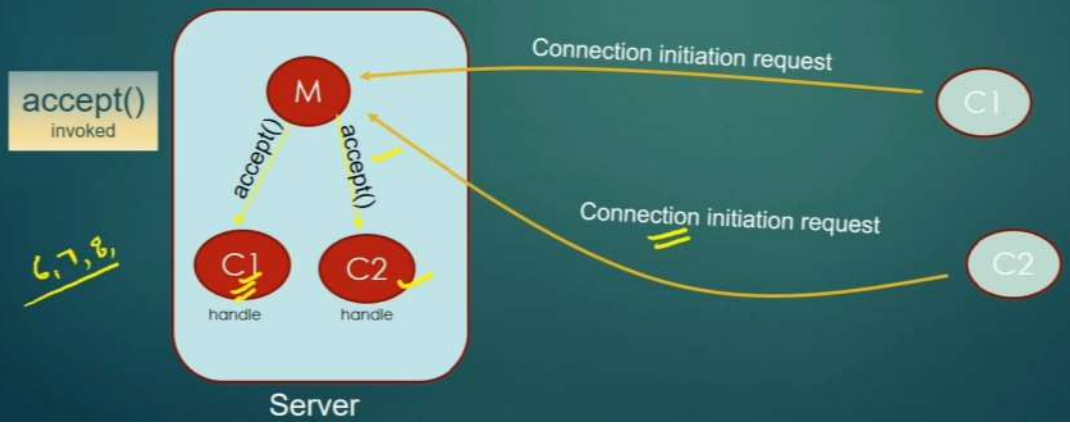
7. In Linux Terminology, handles are called as *"file descriptors"* which are just positive integer numbers. Client handles are called *"communication file descriptors"* Or *"Data Sockets"* and M is called *"Master socket file descriptor"* Or *"Connection socket"*



Linux accept() system call

accept()

- When the server receives new connection initiation request msg from new client, this request goes on Master socket maintained by server. Master socket is activated.
- When Server receives connection initiation request from client, Server invokes the accept() to establish the bidirectional communication
- Return value of accept System call is Client handle Or communication file descriptor
- *accept()* is used only for connection oriented communication, not for connection less communication



UNIX DOMAIN SOCKETS

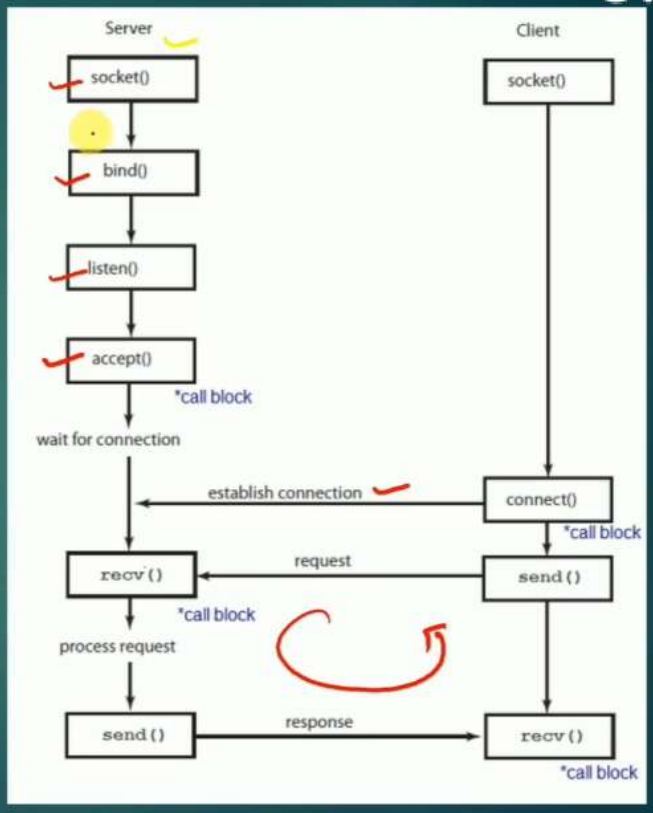
5.1

- Unix Domain Sockets are used for carrying out IPC between two processes running on SAME machine
- We shall discuss the implementation of Unix Domain Sockets wrt to Server and Client processes
- Using UNIX Domain sockets, we can setup STREAM Or DATAGRAM based communication
  - STREAM - When large files need to be moved or copied from one location to another, eg : copying a movie
    ex : continuous flow of bytes, like water flow
  - DATAGRAM - When small units of Data needs to be moved from one process to another within a system

Unix Domain sockets -> code Walk

- Code Walk for Process A (Server)
- Steps :

1. Remove the socket, if already exists
2. Create a Unix socket using *socket()*
3. Specify the socket name
4. Bind the socket using *bind()*
5. listen()
6. accept()
7. Read the data recvd on socket using read()
8. Send back the result using *write()*
9. *close* the data socket
10. *close* the connection socket
11. Remove the socket
12. exit



Generic steps for socket programming

Unix Domain sockets -> Observation

- While Server is servicing the current client, it cannot entertain new client
- This is a drawback of this server design, and we need to alleviate this limitation
- A server can be re-designed to server multiple clients at the same time using the concept of *Multiplexing*

Socket Communication APIs

6.1

- High level Socket Communication Design

Msg type	Client side API	Server side API
Connection initiation request msgs	connect() ⟷	accept() (blocking)
Service request msgs	sendmsg(), sendto(), write()	recvmsg(), recvfrom() read() (blocking)

Multiplexing

- Multiplexing is a mechanism through which the Server process can monitor multiple clients at the same time
- Without Multiplexing, server process can entertain only one client at a time, and cannot entertain other client's requests until it finishes with the current client
- With Multiplexing, Server can entertain multiple connected clients simultaneously

No Multiplexing

Once the current client is serviced by the server, Client has to join the queue right from the last (has to send a fresh connection request) to get another service from the server

Multiplexing

Server can service multiple clients at the same time

Multiplexing -> select

7.1

select()

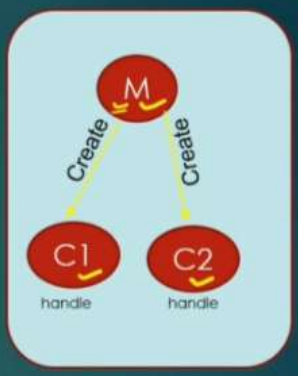
Monitor all Clients activity at the same time

Multiplexing -> select

7.2

- Server-process has to maintain client handles (communication FDs) to carry out communication (data exchange) with connected clients
- In Addition, Server-process has to maintain connection socket Or Master socket FD as well (M) to process new connection req from new clients
- Linux provides an inbuilt Data structure to maintain the set of sockets file descriptors

  fd_set
- select() system call monitor all socket FDs present in *fd_set*



Server

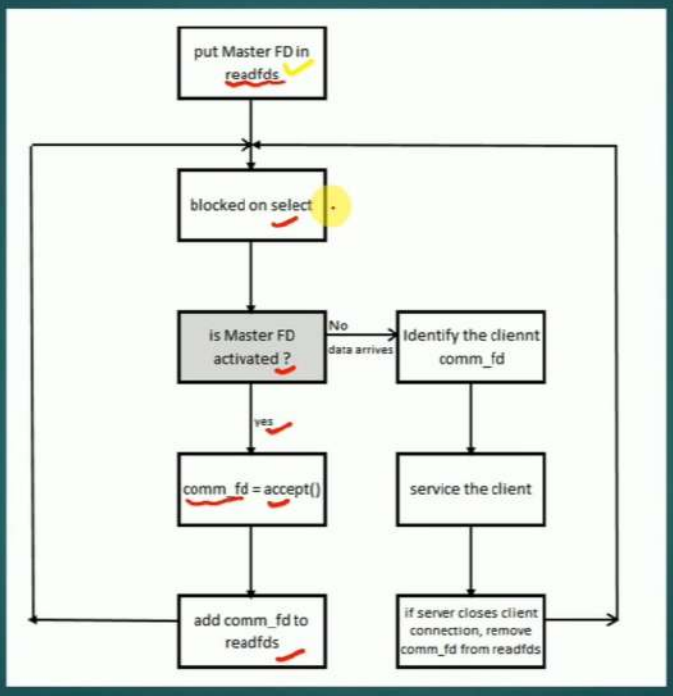
- Let us have a detailed discussion on *select()* system call

Select and accept together in action

7.3

Multiplexed Server process state machine diagram

7.4



Time for a Code walk …

Select and accept together in action

7.5