**FILE TRANSFER PROTOCOL**

**Project Report**

SUBMITTED BY

**SY CS**

**NEHA YADAV 15070121137**

**PRASHANT MALL 15070121146**

**RAJESHWAR SINGH 15070121150**

****

**SYMBIOSIS INSTITUTE OF TECHNOLOGY**

**(A CONSTITUTENT OF SYMBIOSIS INTERNATIONAL UNIVERSITY)**

**Pune - 412115**

**2016-17**

1. **Introduction**
   1. **Problem statement of Proposed system**

In telecommunications, a communication protocol is a system of rules that allow two or more entities of a communications system to transmit information via any kind of variation of a physical quantity. These are the rules or standard that defines the syntax, semantics and synchronization of communication and possible error recovery methods. Protocols may be implemented by hardware, software, or a combination of both.

The File Transfer Protocol (FTP) is a standard network protocol used to transfer computer files between a client and server on a computer network.

Today’s web browsers allow us to download files via FTP from within the browser window. It’s very convenient, and it’s great for downloading a file or two, but the browser-download method does not offer much in the realm of flexibility. We can’t upload, force a transfer mode, or ask the server any questions. But for any sort of web development we need all these functionalities and that’s where FTP comes to our rescue.

This project aims to develop a similar local FTP server to allow file transfers between local client and servers.

* 1. **Data Structure Used**

The given problem statement uses the basic data structure of **Queue** for temporary storage and transfer of files. The queue follows a First in first out approach which enables file transfers to occur in the same manner. The file that is requested first by the client is transferred first. All requested files are maintained in a Queue and transferred accordingly.

* 1. **Purpose of the system**

The FTP system serves for the following purposes:

- To promote sharing of files (computer programs and/or data).

- To encourage indirect or implicit use of remote computers.

- To shield a user from variations in file storage systems among different hosts.

- To transfer data reliably, and efficiently.

* 1. **Project scope**

1. To ensure portability and therefore compatibility.
2. To ensure our system moves with time i.e. allow for maintenance, upgrades and periodic back up by developed and authorized personnel.
3. To program the system using the appropriate database design application, platform and programming languages.
   1. **Definitions/Acronyms**

**FTP (File Transfer Protocol):** standard network protocol used to transfer computer files between a client and server on a computer network.

**TCP/IP (Transmission Control Protocol/Internet Protocol)**: basic communication language or protocol of the Internet. It can also be used as a communications protocol in a private network (either an intranet or an extranet).

**Port:** A port is best envisioned as a little door on the server. Each port number is dedicated to a corresponding application on the server, and the traffic intended for an application. For example, the FTP protocol is handled by ports 20 and 21 by default.

* 1. **Limitations**

1. Cabling and installation was a tedious task.

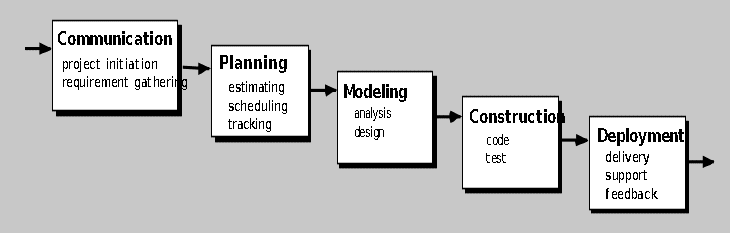
2. Observation was fixed to a specific area.

3. The camera cannot be easily moved to another location.

1. **Project organization**
   1. **Methods, tools and techniques**

**// Hardware and Software tools/Programming Languages**

**PROCESS MODEL USED: WATERFALL**



This model is used as it is easy to implement for small projects.

**Hardware**

* Pentium® IV Recommended
* 128 MB RAM (256 MB RAM recommended)
* 100 MB free hard drive space (250 MB free hard drive space recommended)
* High Colour display adapter at 800x600 resolution (High Colour display adapter at 1024x768 resolution recommended)
* Dial-Up Internet Connection (Broadband Highly Recommended, Cable Modem or DSL)

**Software**

* Windows® 7 (Seven), Windows Vista, Windows Server 2003/2008, Windows XP, Windows 2000/NT (Windows® 7, Vista or XP recommended), Windows 8, Windows 10, Ubuntu or any Linux Operating System
* Microsoft® Internet Explorer™ 5.0.0 (Microsoft® Internet Explorer™ 6.0.0 recommended)

**Programming Language used: C**

1. **Requirement gathering, refinement and analysis**

**Entities:** 1) client 2) server

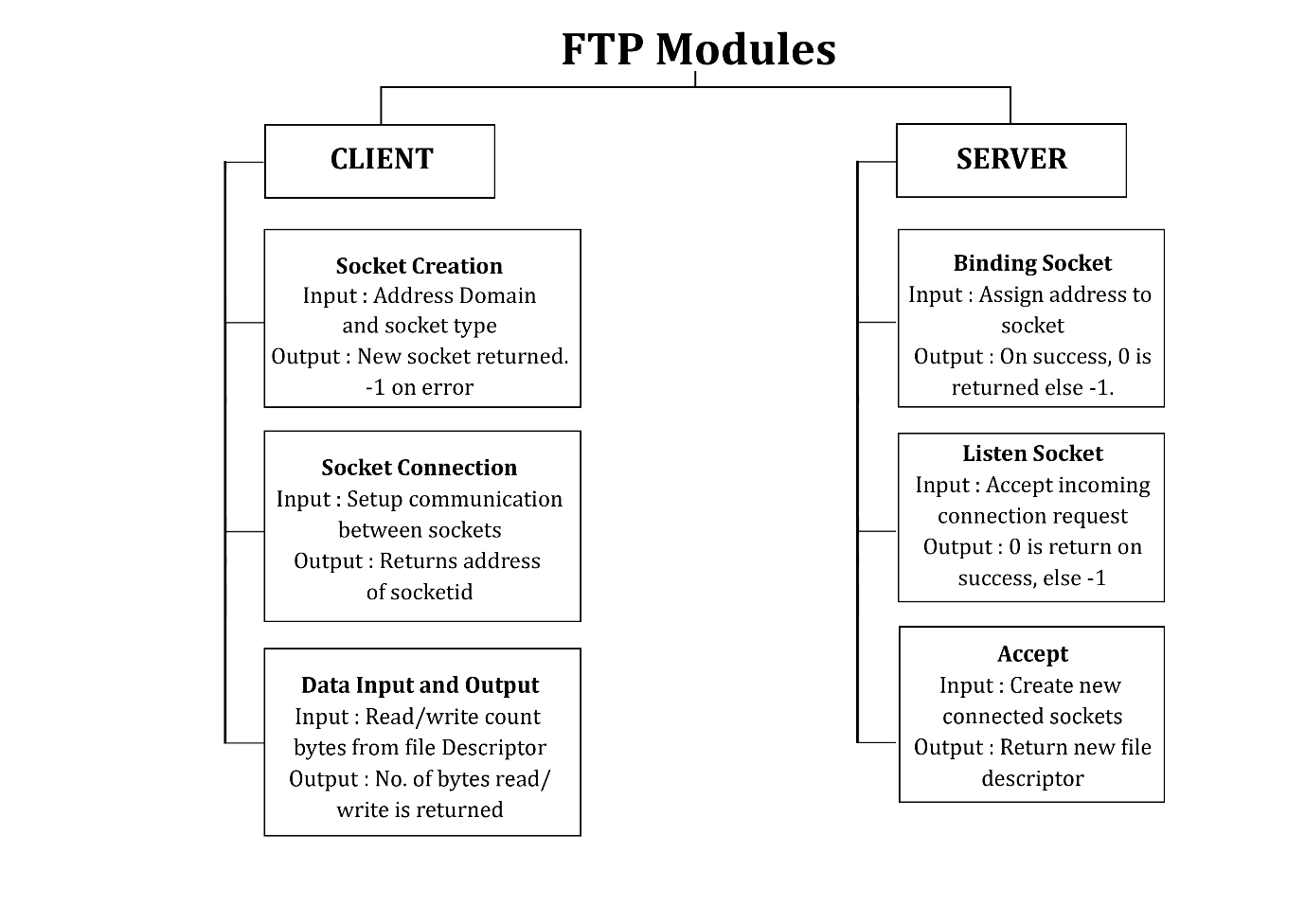
**Major functions**

**Client**

1. Create a socket with the socket () system call
2. Connect the socket to the address of the server using the connect () system call
3. Send and receive data. There are several ways to do this, but the simplest is to use the read () and write () system calls.

**Server**

1. Create a socket with the socket () system call
2. Bind the socket to an address using the bind () system call. For a server socket on the Internet, an address consists of a port number on the host machine.
3. Listen for connections with the listen () system call
4. Accept a connection with the accept () system call. This call typically blocks until a client connects with the server.
5. Send and receive data



* 1. **Requirement 1: SOCKET CREATION**
     1. Refinement of Socket creation:

**Input:**  specify the *address domain* and the *socket type*

**Output:** file descriptor for the new socket is returned using **socket ()**. On error, -1 is returned, and *errno* is set appropriately.

* 1. **Requirement 2: SOCKET CONNECTION**
     1. Refinement of Socket connection:

**Input:** Two processes communicate with each other when their sockets are of the same type and in the same domain.

**Output: connect** () system call connects the socket referred to by the file descriptor *sockfd* to the address specified by *addr*

* 1. **Requirement 3: Data input and output** 
     1. Refinement of read

**Input:** attempt is made to read up *count* bytes from file descriptor *fd* into the buffer starting at *buf using* ***read()*.**

**Output:** the number of bytes read is returned (zero indicates end of file), and the file position is advanced by this number.

* + 1. Refinement of write

**Input:** writes up to *count* bytes from the buffer pointed *buf* to the file referred to by the file descriptor *fd using* ***write ()***

**Output:**the number of bytes written is returned (zero indicates nothing was written). On error, -1 is returned, and *errno* is set appropriately.

* 1. **Requirement 4: BIND SOCKET**

**3.4.1** Refinement of bind

**Input: bind** () assigns the address specified by *addr* to the socket referred to by the file descriptor *sockfd*.

**Output:** On success, zero is returned. On error, -1 is returned, and *errno* is set appropriately.

* 1. **Requirement 5: LISTEN SOCKET**
     1. Refinement of listen

**Input:** accept incoming connection requests using **accept ().**

**Output:** zero is returned. On error, -1 is returned.

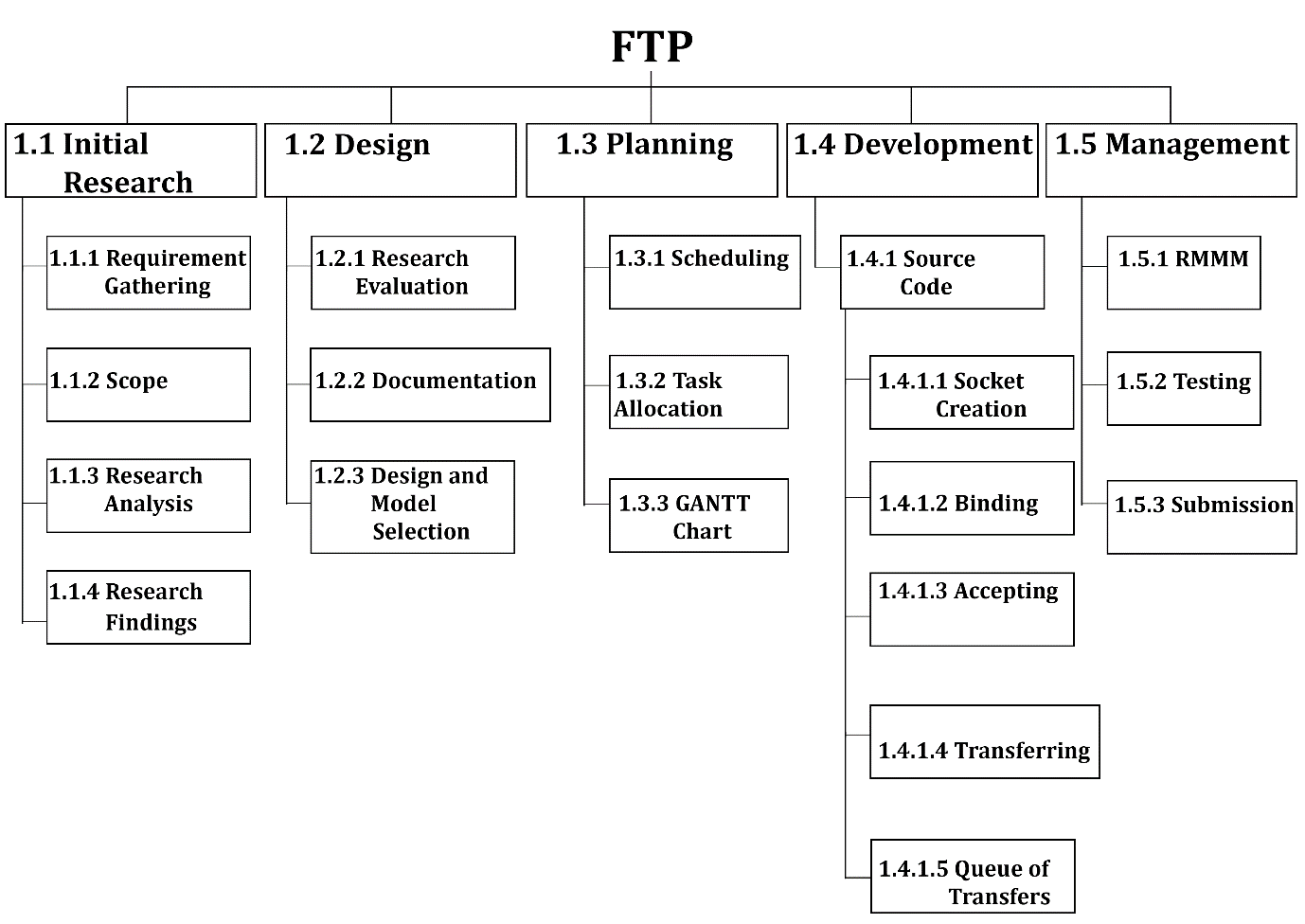
**3.6 Requirement 6: ACCEPT**

**3.6.1** Refinement of accept

**Input:** extracts the first connection request on the queue of pending connections for the listening socket, *sockfd*, and creates a new connected socket

**Output:** returns a new file descriptor referring to that socket.

1. **Project scheduling**
   1. **Functional decomposition (WBS)**



* 1. **Roles and Responsibilities (staff allocation such as sr. no., task, Team member involved, responsibilities)**

|  |  |  |  |
| --- | --- | --- | --- |
| **Sr. No.** | **Phase of SDLC** | **Team Members Involved** | **Responsibilities** |
| 1 | Requirement gathering | Prashant, Neha | Prepare initial report |
| 2 | Analysis | Rajeshwar, Neha | Project plan with cost and effort estimation |
| 3 | Design | Prashant, Rajeshwar, Neha | Case diagrams with specifications, data flow diagram, activity diagram, documentation |
| 4 | Implementation | Prashant, Rajeshwar, Neha | Develop source code |
| 5 | Testing | Prashant, Rajeshwar | Prepare testing document |
| 6 | Deployment | Prashant, Rajeshwar, Neha | Deliver project |

**4.3 Task Duration**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sr. No.** | **Task** | **Days** | **Start Head** | **Finish Head** |
| **1** | **Scope** | **5** | **13/01/17** | **17/01/17** |
| 1.1 | Determine the Project Scope | 1 | 13/01/17 | 13/01/17 |
| 1.2 | Define Preliminary Resources | 2 | 14/01/17 | 15/01/17 |
| 1.3 | Establish Project Goals | 2 | 16/01/17 | 17/01/17 |
| **2** | **Analysis / Software Requirements** | **20** | **20/01/17** | **29/01/17** |
| 2.1 | Conduct Project Requirement Analysis | 7 | 20/01/17 | 26/01/17 |
| 2.2 | Draft Preliminary Software Specifications | 2 | 27/01/17 | 28/01/17 |
| 2.3 | Obtain approval to proceed further | 1 | 29/01/17 | 29/01/17 |
| **3** | **Design** | **15** | **01/02/17** | **15/02/17** |
| 3.1 | Develop Function Specification | 6 | 01/02/17 | 06/02/17 |
| 3.2 | Develop Initial Documentation | 4 | 07/02/17 | 10/02/17 |
| 3.3 | Review Functional Specification | 3 | 11/02/17 | 13/02/17 |
| 3.4 | Obtain Approval to proceed further | 2 | 14/02/17 | 15/02/17 |
| **4** | **Implementation and Coding** | **32** | **17/02/07** | **20/03/17** |
| **5** | **Review Functional Specification** | **3** | **21/03/17** | **23/03/17** |
| **6** | **Identify Modules and Testing** | **8** | **31/03/17** | **7/04/17** |

* 1. **RMMM table**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **SR no.** | **Risk** | **Description** | **Probability** | **Impact** | **RMMM** |
| 1 | Compatibility Issue | May have compatibility issues with certain versions of OS | 49 | High | R1 |
| 2 | Security Breaches | Socket connection may not be secure and data gets compromised | 10 | Medium | R2 |
| 3 | Data Corruption | Data may get corrupted while transfer. | 15 | Medium | R3 |
| 4 | Connection Loss | Connection breaks and data transfer gets interrupted | 15 | Medium | R4 |
| 5 | Storage limitation | Insufficient storage at receiver end may stop the transfer of data/file. | 5 | Low | R5 |

**R1:**

**Mitigation**

1. Keeping minimal compatibility of the software.

**Monitoring**

1. Check for regular updates of OS.

**Maintenance**

1. Implementing new updates.

**R2:**

**Mitigation**

1. Encryption of files while sending and receiving.

**Monitoring**

1. Changing and checking encryption pattern.
2. Clean up of temporary files.

**Maintenance**

* 1. Timely improvement of security.

**R3:**

**Mitigation**

1. Detection of data corruption at receiver end.

**Monitoring**

1. Taking feedback after every transfer of data/file.

**Maintenance**

1. Resending data/file in case of negative feedback.

**R4:**

**Mitigation**

1. Establish secure network connection.

**Monitoring**

1. Keep track of network connection.

**Maintenance**

1. Re-establishing network connection and send data/file.

**R5:**

**Mitigation**

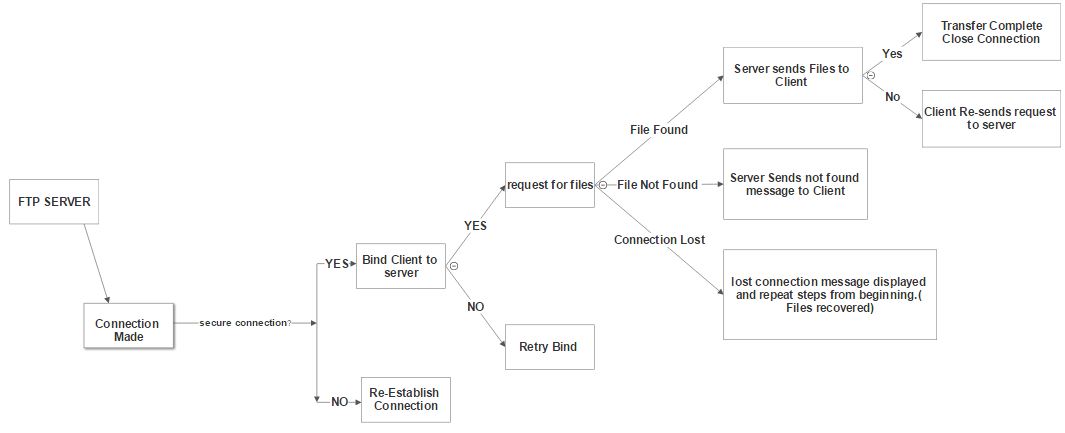
1. Checking storage availability of receiver system.

**Monitoring**

1. Taking feedback from receiver.

**Maintenance**

1. Send notification in case of insufficient storage.
2. **Non-Functional Requirements**
3. **Reliability:** Reliability reflects the capacity of the software to maintain its performance over the time. It implies how well the system performs in peak hours. A robust system is one which has the capacity to handle the bugs without failure i.e. how effortlessly it handles the bugs. The failure rate in FTP server should be least or negligible as the system is supposed to be reliable. Reliability of the system depends on the failure free transfers and how fast the system is able to recover from the failure.
4. **Security:** Security is the feature of the system which ensures that system must be protected from the unintentional or malignant harm; unauthorized access to the data is not permissible. For the safety purpose the data must be backed up and stored in a secure location. In FTP Server the Client must be able to send or receive the information to or from the server and client in an encrypted way. Security must stick to some standard and plans. The security is significant subject of FTP as client is more worried about the security of the files being transferred. The information kept in the system ought to be precise and complete.
5. **Portability:** It is concerned with the how well the software will work in distinctive environment. The FTP application must have the capacity to operate on any gadget i.e. hand held gadgets or desktops or laptops without failure the change in the environment ought not to hamper the operability of the software.
6. **Maintainability**: Maintainability implies the ability to restore your software to the point when failure occurred. The ability to recoup quickly from a system failure depends not simply on having backup of the data, also on having a predefined plan for recuperating that information. The FTP server must have the capacity to recover itself from the failures which may happen due to connection problem with the client or because of some other reason.
7. **Decision Tree**



Decision tree shows the sequence of events that will be followed pertaining to a particular decision of taken in a program.

The edges of a decision tree represent conditions and the leaf nodes represent the

actions to be performed depending on the outcome of testing the condition.

It gives us an idea about what actions will be performed if we take a particular decision in the system.

1. **Project Size and Cost Estimation**

**Method used:** COCOMO Based estimation

**Estimated LOC**

Optimistic: 700

Most Likely: 900

Pessimistic: 1200

LOC= 700+4x900+1200

6

=917 LOC =**0.917 KLOC**

**COCOMO Based Estimation**

**Project type:** Semidetached (as developers do not have much experience in the development of similar project)

For semidetached a1=3, a2=1.12, b1=2.5, b2=0.35

**Effort= a1x(KLOC)^a2 PM**

**=**3x (0.917) ^1.12 =2.7225= **3 PM**

**Tdev (time for development) = b1x(Effort)^b2 Months**

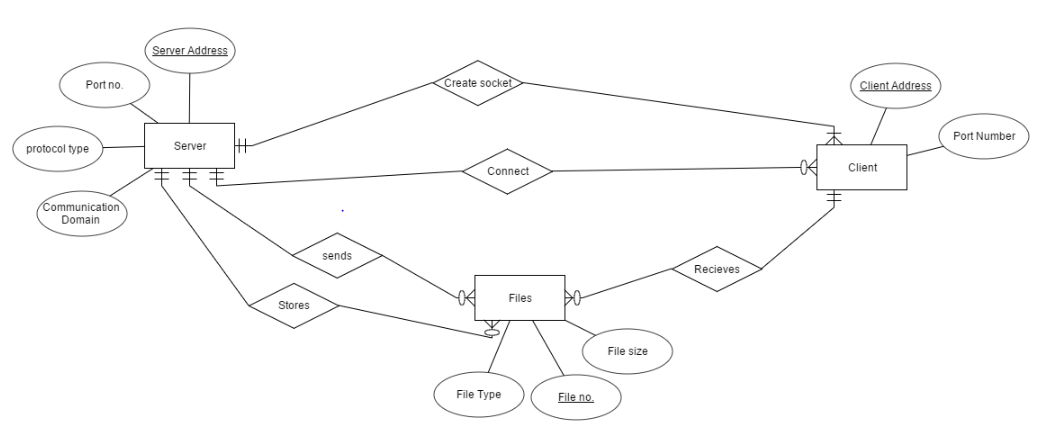
=2.5x (3) ^0.35 = 3.6723 =**3.7 Months**

Assume cost per KLOC=10$

**Cost=** 10x KLOC $

=10x0.917= **9.17$**

1. **Project Design**
   1. **Entity Relationship Diagram**



Description:

* **Entities:** 1) Client 2) Server 3) Files
* **Attributes of Server:** 
  1. Server address: Uniquely identifies the IP address of the server
  2. Port number: Identifies the port number of server through which connection is established.
  3. Protocol type: Identifies type of protocol used for transfer TCP/UDP
  4. Communication domain: PF\_INET, IPv4 protocols, Internet addresses

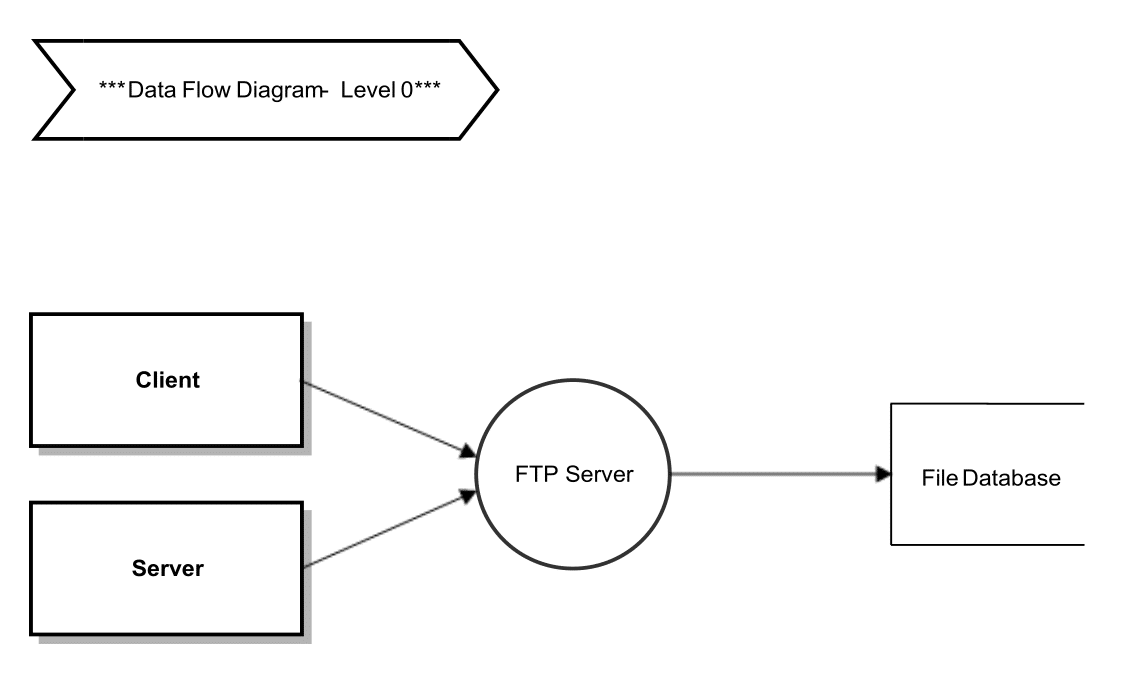
PF\_UNIX, Local communication, File addresses

* **Attributes of client:**
  1. Client address: Uniquely identifies the IP address of the client
  2. Port number: Identifies the port number of server through which connection is established.
* **Attributes of Files:**

1. File no.: Uniquely identifies the number of file being transferred
2. File Size: the size of file in MB or KB
3. File Type: identifies whether file is an image or word document or text file etc.

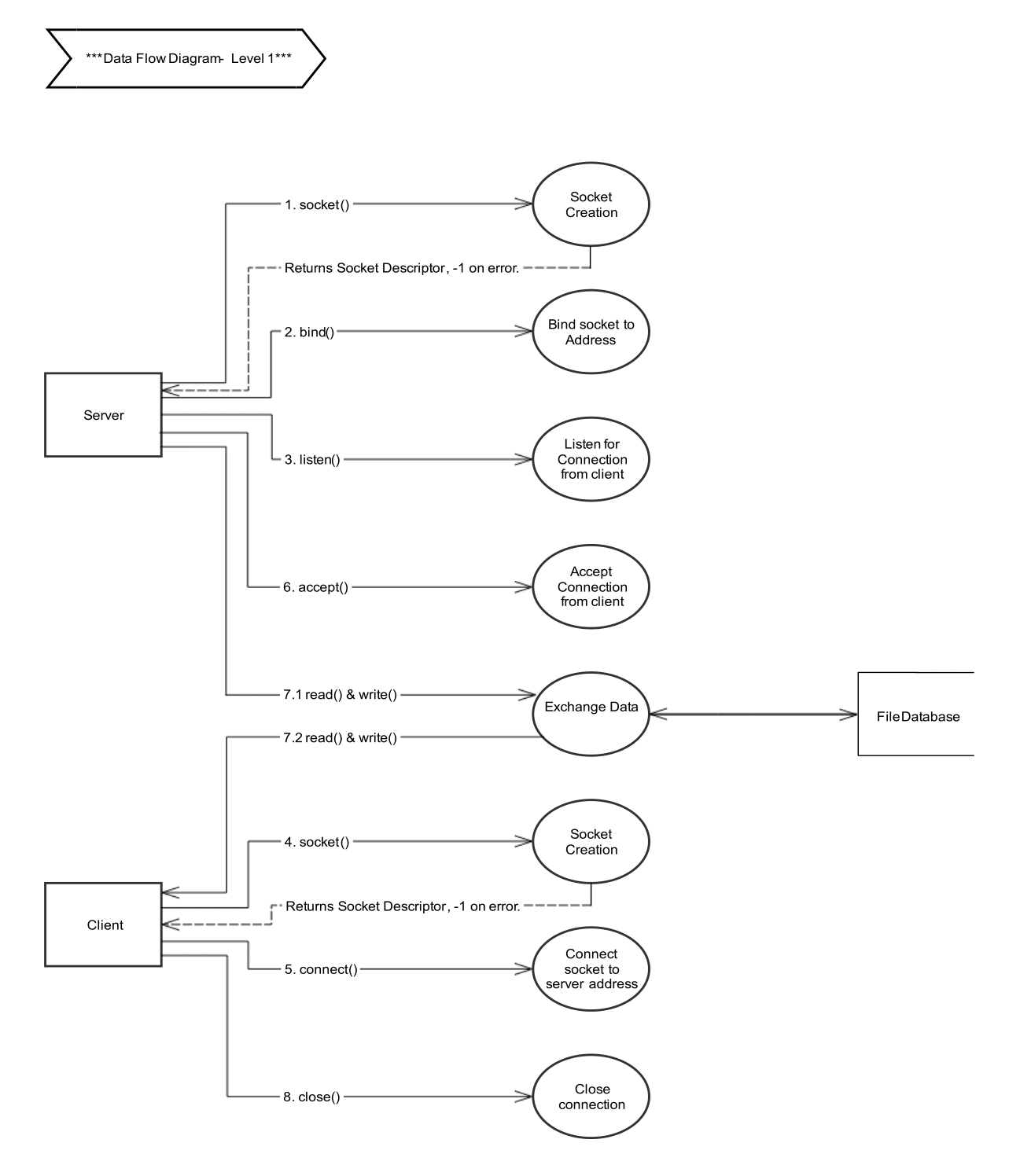
**Relationships:**

1. Create Socket: Server creates socket (modality= mandatory, cardinality= 1:M) to be detected and connected to by many clients at a time. Client on the other hand can connect to the socket of one and only one server.
2. Connect: Server and Client establish a secure Connection. (modality= mandatory for client but optional for server to connect, cardinality= 1:M)
3. Send: Server may or may not send files but files must be sent from the server (modality: optional for server mandatory for files, Cardinality:1:M)
4. Receive: Client may or may not receive a file but files have to be received by the client. (Modality= optional for client mandatory for files, cardinality= M:1)
5. Store: Server stores multiple files at a time in database (Modality: optional for server mandatory for files, Cardinality= 1:M)
   1. **Data Flow Diagram**
      1. **Level 0**



Most interposes communication uses the *client server model*. These terms refer to the two processes which will be communicating with each other. One of the two processes, the *client*, connects to the other process, the *server*, typically to make a request for information. A good analogy is a person who makes a phone call to another person. After a connection is established transfer of data takes place.

* + 1. **Level 1**

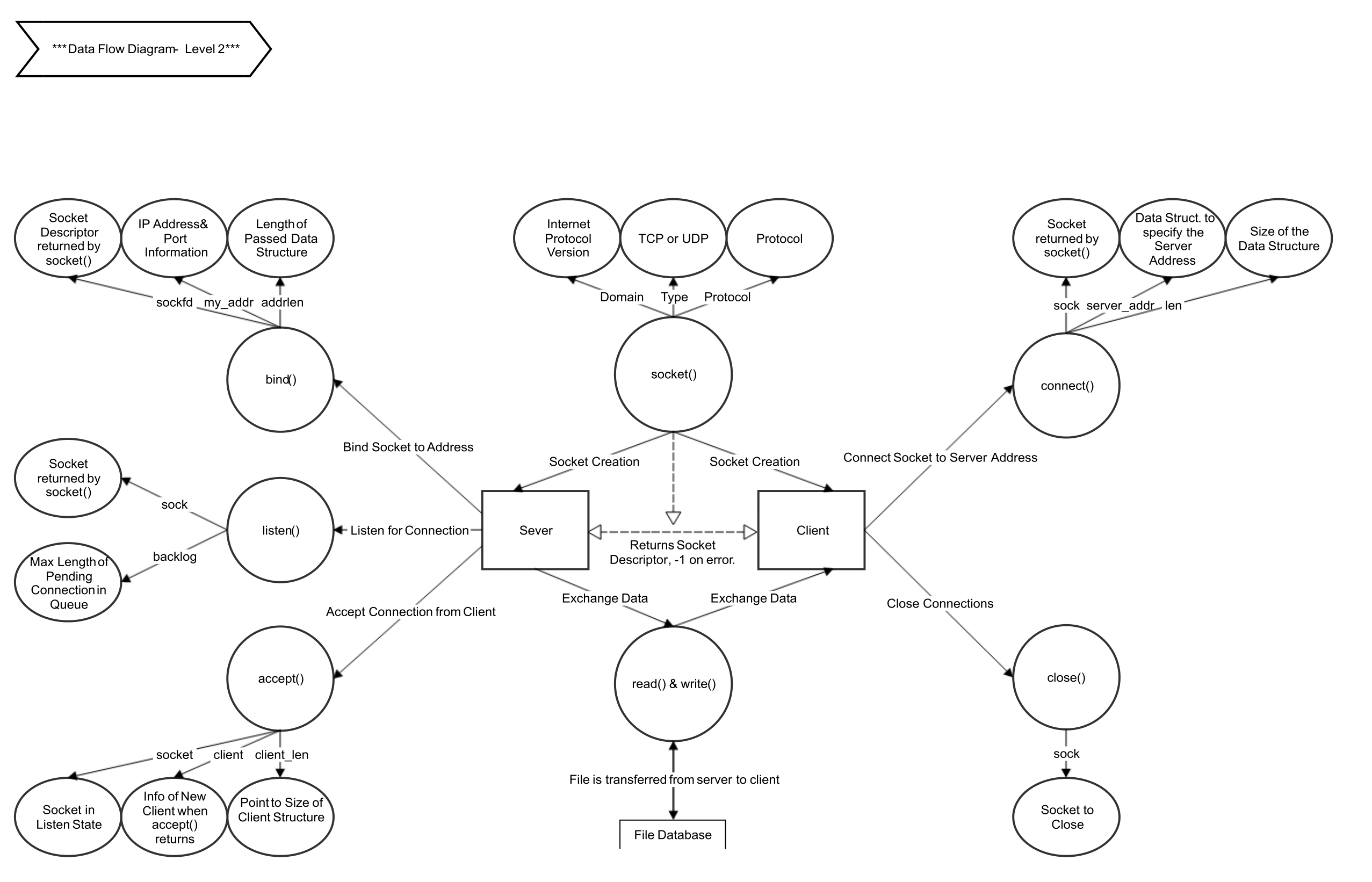


The steps involved in establishing a socket on the *client* side are as follows:

1. Create a socket with the socket() system call
2. Connect the socket to the address of the server using the connect() system call
3. Send and receive data. There are a number of ways to do this, but the simplest is to use the read() and write() system calls.

The steps involved in establishing a socket on the *server* side are as follows:

1. Create a socket with the socket() system call
2. Bind the socket to an address using the bind() system call. For a server socket on the Internet, an address consists of a port number on the host machine.
3. Listen for connections with the listen() system call
4. Accept a connection with the accept() system call. This call typically blocks until a client connects with the server.
5. Send and receive data
   * 1. **Level 2**



* 1. **Socket Creation**

**socket**() creates an endpoint for communication and returns a descriptor. It specifies the address domain and socket type. In case of an error it returns -1.

* 1. **Socket Binding**

When a socket is created with **socket**(), it exists in a name space (address family) but has no address assigned to it. **bind**() assigns the address specified by *addr* to the socket referred to by the file descriptor *sockfd*. *addrlen* specifies the size, in bytes, of the address structure pointed to by *addr*.

* 1. **Listening State**

**listen**() marks the socket referred to by *sockfd* as a passive socket, that is, as a socket that will be used to accept incoming connection requests using **accept**().

The *backlog* argument defines the maximum length to which the queue of pending connections for *sockfd* may grow. If a connection request arrives when the queue is full, the client may receive an error with an indication of **ECONNREFUSED.**

* 1. **Connect to Server**

The **connect**() system call connects the socket referred to by the file descriptor *sockfd* to the address specified by *addr*. The*addrlen* argument specifies the size of *addr*. The format of the address in *addr* is determined by the address space of the socket*sockfd.*

* 1. **Accept Connection from Client**

The **accept**() system call is used with connection-based socket types. It extracts the first connection request on the queue of pending connections for the listening socket, *sockfd*, creates a new connected socket, and returns a new file descriptor referring to that socket. The newly created socket is not in the listening state. The original socket*sockfd* is unaffected by this call.

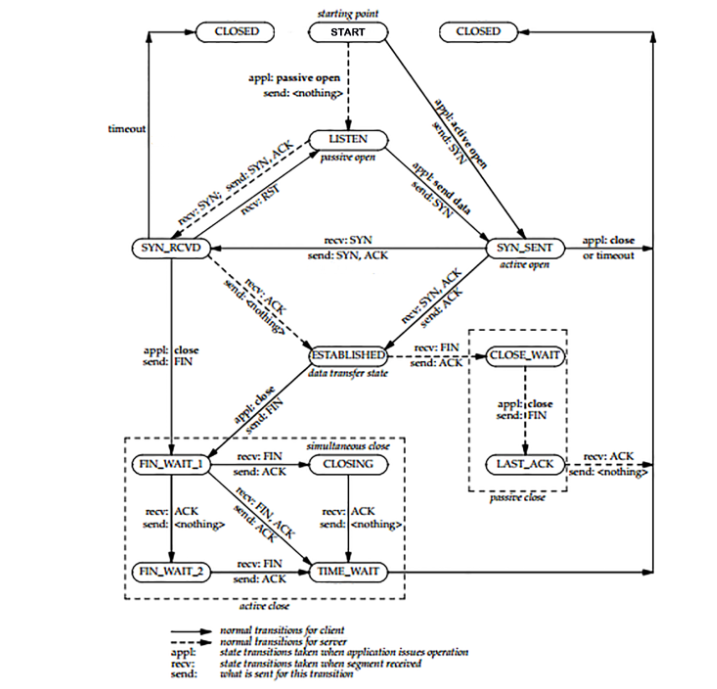
* 1. **Data Transfer**

**read**() attempts to read up to *count* bytes from file descriptor *fd* into the buffer starting at *buf*.

**write**() writes up to *count* bytes from the buffer pointed *buf* to the file referred to by the file descriptor *fd*.

* 1. **Close Connection**

**close()** system call terminates the connection with the server.

* 1. **State Transition Diagram**
* A connection progresses through a series of states during its lifetime. The states are: LISTEN, SYN-SENT, SYNRECEIVED, ESTABLISHED, FIN-WAIT-1, FIN-WAIT-2, CLOSE-WAIT, CLOSING, LAST-ACK, TIME-WAIT, and the fictional state CLOSED. CLOSED is fictional because it represents the state when there is no TCB, and therefore, no connection.

Briefly the meanings of the states are:

* LISTEN represents waiting for a connection request from any remote TCP and port.
* SYN-SENT represents waiting for a matching connection request after having sent a connection request.
* SYN-RECEIVED represents waiting for a confirming connection request acknowledgment after having both received and sent a connection request.
* ESTABLISHED represents an open connection, data received can be delivered to the user. The normal state for the data transfer phase of the connection.
* FIN-WAIT-1 represents waiting for a connection termination request from the remote TCP, or an acknowledgment of the connection termination request previously sent.
* FIN-WAIT-2 represents waiting for a connection termination request from the remote TCP.
* CLOSE-WAIT represents waiting for a connection termination request from the local user.
* CLOSING represents waiting for a connection termination request acknowledgment from the remote TCP.
* LAST-ACK represents waiting for an acknowledgment of the connection termination request previously sent to the remote TCP (which includes an acknowledgment of its connection termination request).
* TIME-WAIT represents waiting for enough time to pass to be sure the remote TCP received the acknowledgment of its connection termination request.
* CLOSED represents no connection state at all.

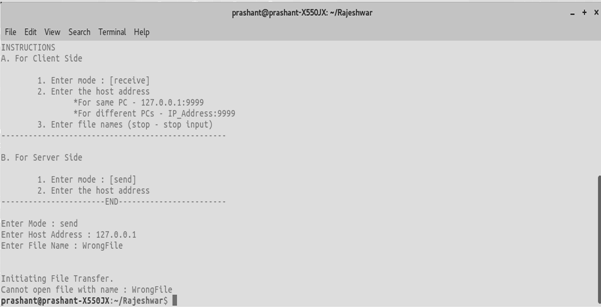
A FTP connection progresses from one state to another in response to events. The events are the user calls, OPEN, SEND, RECEIVE, CLOSE, ABORT, and STATUS; the incoming segments, particularly those containing the SYN, ACK, RST and FIN flags; and timeouts.

1. **Test Cases**
   1. **Wrong file name or File doesn’t exist**

Expected file not found in the source directory.

Prints the error message “Cannot open file with name: <Filename>”

The file must exist in the program execution directory.

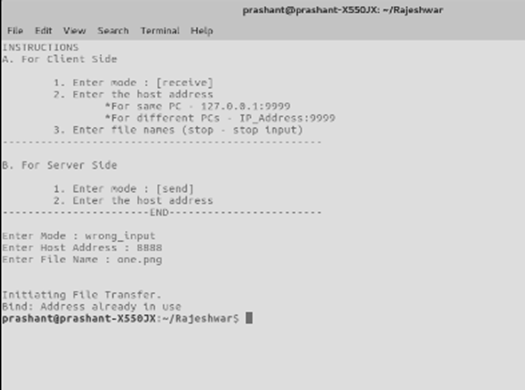
****

* 1. **Address already in use**

When server tries to establish a new connection with a client on which binding is already done, then an error message is prompt on screen

Message: “Bind: Address already in use”

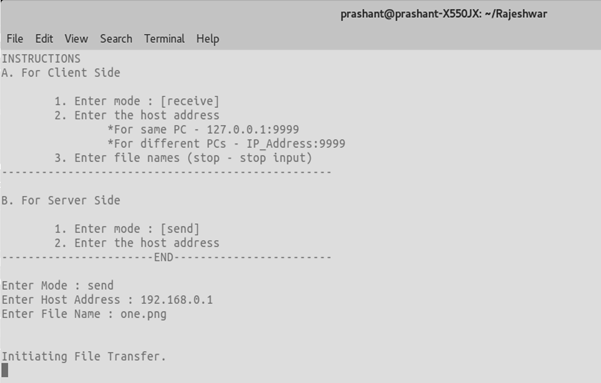
The server-client must close all the previous connections before establishing a new one.



* 1. **Wrong IP address (Program halts after initiating transfer)**

When wrong IP address is entered by user on server side then the connection is never established and server goes in infinite waiting state as there is no client IP to bind to.

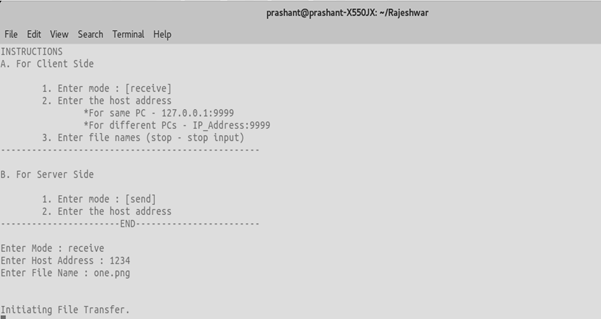
The user must verify and then input Client IP Address on Server to establish proper connection and initiate file transfer.



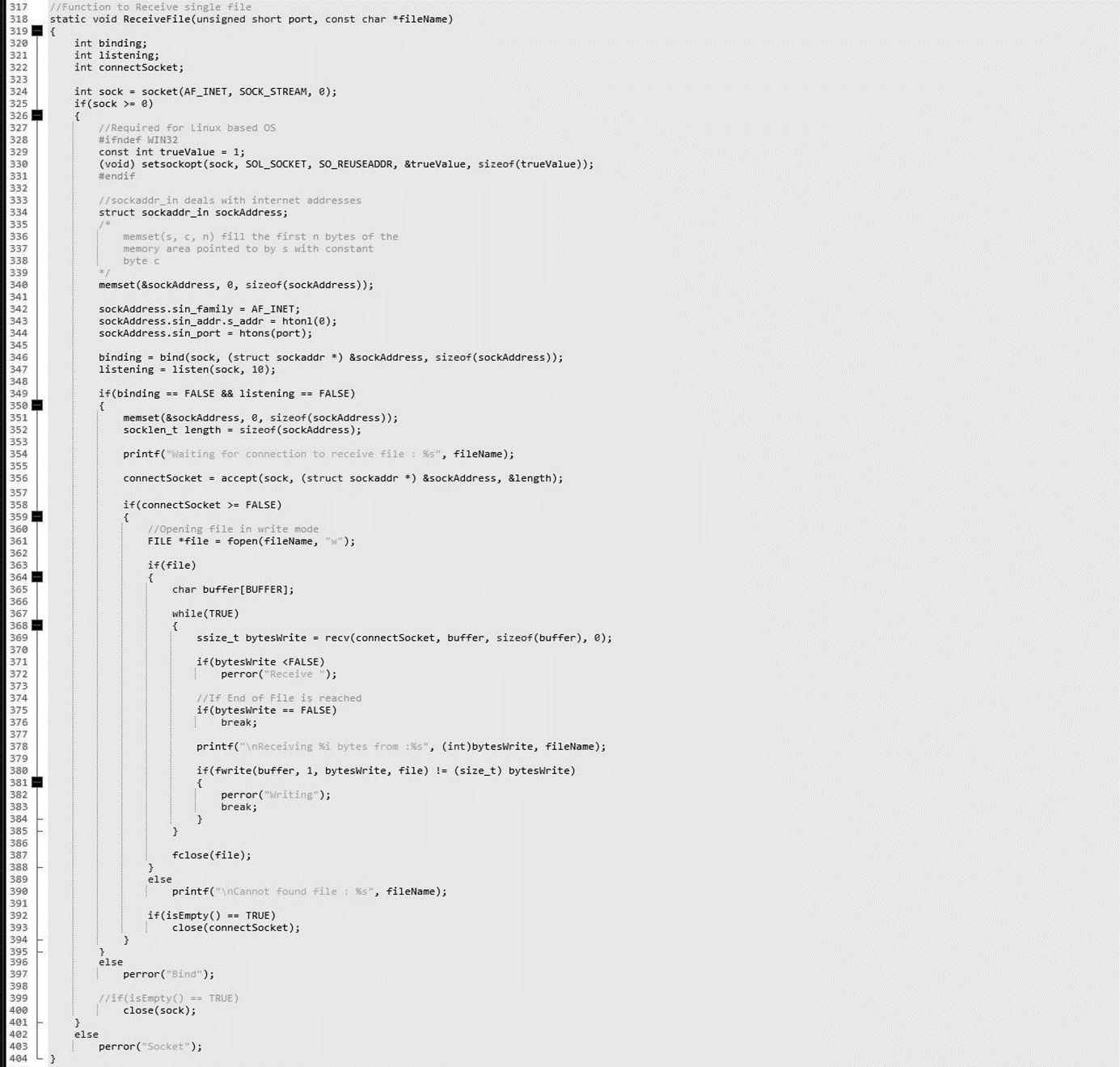
* 1. **Wrong Port Number (Program halts after initiating transfer)**

When wrong Port Number is entered by user on client side then the server never finds the client to bind with.

The user must verify and then input Client Port Number to establish proper connection and initiate file transfer.



1. **Implementation** 
   1. **Receive Function**

****

* 1. **Send Function**

