British Columbia Institute of Technology

Client-Server Design

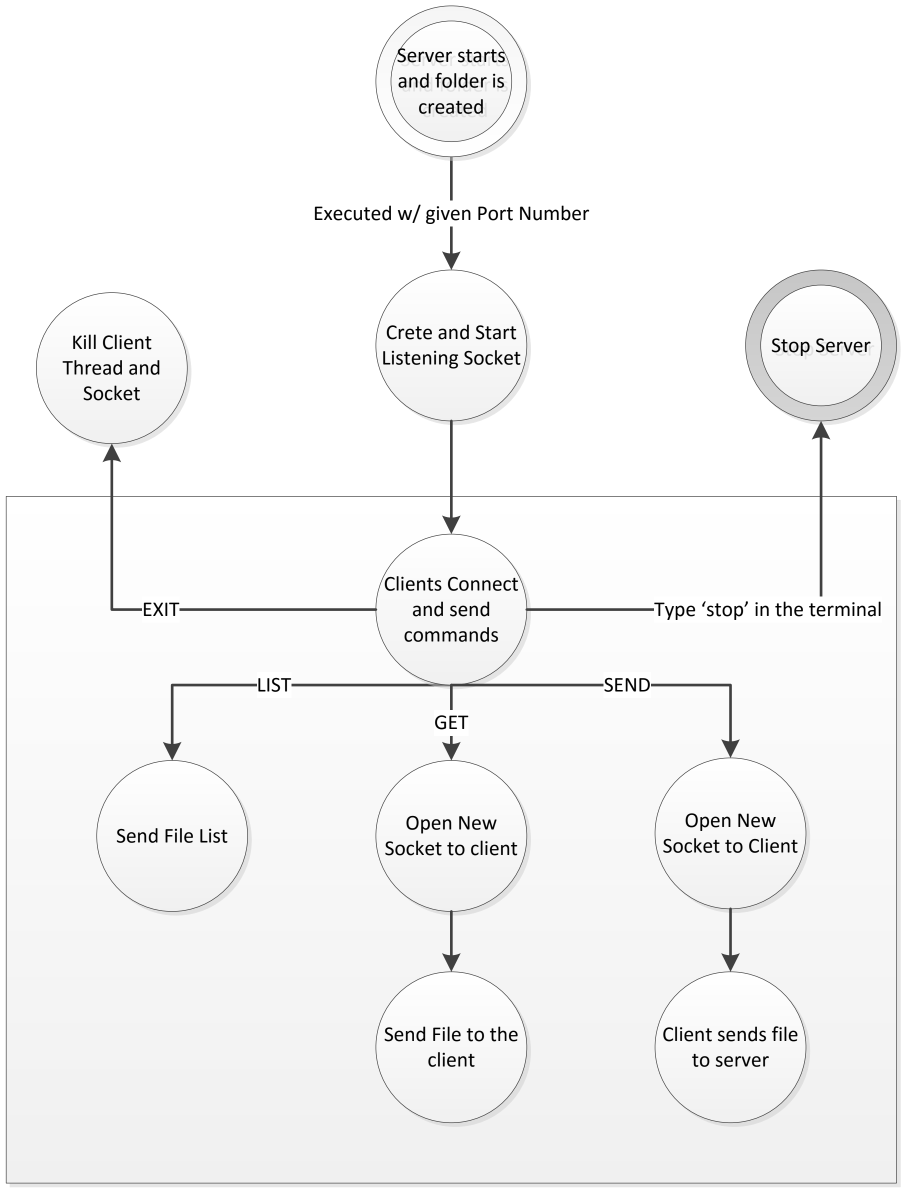
Assignment 1 - COMP 7005

Manuel Gonzales

A00866174

October 05, 2015

SERVER DIAGRAM


Server starts and folder is created
Executed w/ given Port Number
Crete and Start Listening Socket
Kill Client Thread and Socket
Stop Server
Clients Connect and send commands
EXIT
Type 'stop' in the terminal
LIST
GET
SEND
Send File List
Open New Socket to client
Open New Socket to Client
Send File to the client
Client sends file to server

Pseudocode (to be implemented in Ruby)

```
Create and Start Listening Socket
{
    Create Stream Socket and bind it to the selected port.
    Start Listening for new Connections
}

Thread to receive user inputs
{
    If (input == stop)
        Stop Server()
}

Thread to receive commands from clients(for each)
{
    While ()
    {
        If (commands == exit)
            Exit ()
        If (commands == list)
            List ()
        If (commands == get)
            SendFile()
        If (commands == send)
            GetFile()
    }
}
```

```
Exit
{
        Send Exit message to Client
        Close the socket
        Stop client thread
}

Stop Server()
{
        Close listening Socket
        Notify Clients
        Close sockets
        Exit the Application
}

List()
{
        Check for all the files in the ‘files’ folder
        Send List to Client
}

SendFile()
{
        Open New Socket to the client
        Send Complete File to the Client
        Close Socket
}
```

```
GetFile()
{
    Open New Socket to the client
    Receive Complete File from the Client and store it in the File Folders.
    Close Socket
}
```

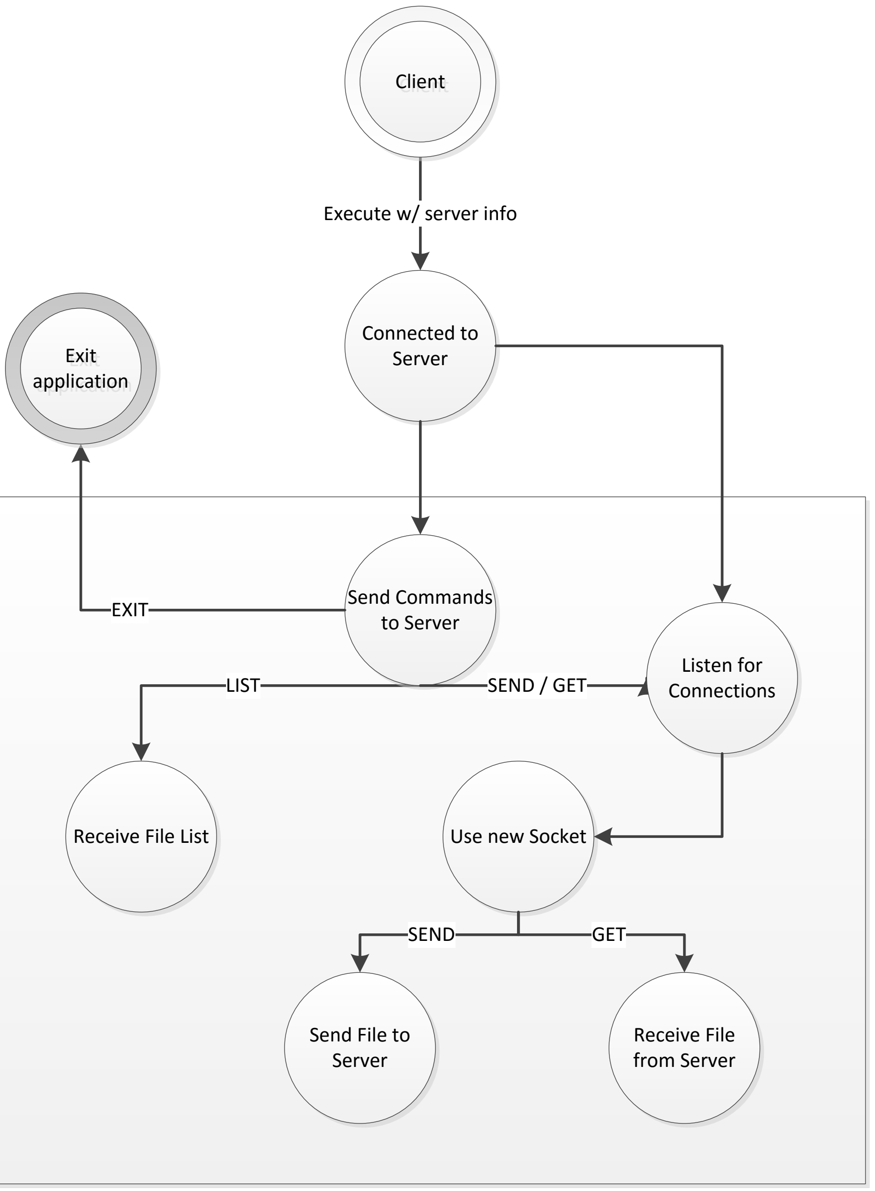
Some Constraints:

The server only shows the content of its folder called ‘files’ that will be created when it is running if it does not exist.

Files will not be overwritten so files with the same name will not be received from clients.

Files are case sensitive.

CLIENT DIAGRAM



Pseudocode (to be implemented in Ruby)

```
Connected to server
{
    Use TCP Socket to stablish a connection to the server.
}

Thread to receive user inputs
{
    Read input from user and send it straight to the server.
}

Thread to receive server commands
    While ()
    {
        If (commands == exit)
            Exit Application ()
        Else If (commands == receiving)
            SendFile()
        Else If (commands == sending)
            GetFile()
        Else
            Show it on terminal
    }
}
```

```
Listen For Connections
{
    Create Stream Socket and bind it to the selected port.
    Start Listening for new Connections
}

Exit Application
{
    Send Exit message to Server
    Get Response and Close the Socket
    Exit application
}

Send File()
{
    Use new socket from the server
    Check if file already exists.
    Send Complete File to the server
    Close Socket
}

GetFile()
{
    Use new socket from the server
    Check if file already exists.
    Receive Complete File from the server and store it in the current folder.
    Close Socket
}
```

Some Constraints:

The client will send files from the current directory and it will store received files in there as well

Files will not be overwritten so files with the same name as existing ones will not be requested from the server

Files are case sensitive.