Download Server / Client

> Server written by: Evan Buss

> Client written by: Evan Buss

> CSC328 - Network Programming

The application is a download server and client written in Java. The client will connect to the server and have the ability to traverse directories and download files found.

The server allows anonymous connections without username/password. To get started compile the files and run them with the proper command-line arguments.

Compilation

The download and server files need to be compiled from .java files to .class files in order to be run.

Client

```
javac Client.java
```

Server

```
javac Server.java
```

Both

```
javac Client.java Server.java
```

Execution

The server program should be executed before the client.

Client

The client receives the host and optional port number as command line arguments.

Arguments

- `<host>` (required) - the IP address or server name to connect to
- `[port]` (optional) - the port number to connect to on the host

By default, the client will attempt to connect to port 5001 of `<host>` if you do not specify a port.

```
java Client <host> [port]
```

Server

Argument

- `[port]` (optional) - the port number that the server should be bound to

By default, the server will attempt to bind to port 5001 if a specific port number is not specified.

`java Server [port]`

Design Overview

The project consists of two parts. The download server and the client. The client serves to test the download server. The server program allows a connected client to navigate its directories and download files.

The server is implemented with with a concurrent multi-threaded model. This enables multiple clients to be connected at the same time without without the server blocking requests.

The server and client use a TCP connection to send information. This is important because we need the connection when downloading files to be reliable.

Server Protocol

HELP (Client Only)

- Lists the valid client commands to the user
- Does not send any data to the server

BYE

- Tells the server to close connection and exits the client program afterwards
- **Protocol**
  - The client sends a string terminated by a return character.
  - The server reads the string until the newline character and checks its validity
- **Response**: Server Closes Connection

PWD

- Server sends the client its current working directory path.
- Client sets the path indicator in the shell prompt to the updated directory value
- **Response**: Directory path string.
- **Data Validation**
  - The client knows it has received the entire path string when it encounters a newline character.
  - The server uses Java's println() method to send the path string to the client with a newline character ('\n') at the end via an output stream.
  - The client uses Java's readLine() method on the input stream to read until it encounters a newline ('\n') character.

    - If the methods do not find the newline character, they fail.

DIR

- Server sends a list of all files and folders in the current working directory.
- The client displays this list to the user.
- **Design Decisions**
    - Each file or folder encountered is sent back written as three pieces of data
        - Descriptor that says if the given name is a file or folder
        - The size of the file or folder
        - The name of the file or folder
    - These pieces of data are concatenated in a single string on the server side
        - Each token (or piece of data) is ended with a "#" character.
    - The client is responsible for parsing the tokens and formatting however it wishes.
    - This decision allows the server to send a lot of detail about each item without wasting data on formatting characters like tabs, newlines, and spaces
    - All files in the directory are listed before the folders are listed so the user can easily tell what can be downloaded and what cannot (files can be downloaded...folders cannot).
    - The client currently parses the string in groups of three and displays each in a single formatted line.
    - If the directory is empty, the server sends back an `EMPTY` error code. The client checks for that and outputs the error if present.
- **Data Validation**
    - Because the file details are formatted into a single string, the same validation method can be used that was used above in the `PWD` command.

CD <absolute path/relative path/..>

- Server switches client's current working directory to the given directory

- Only accepts a single argument at a time. If there are multiple words, they are considered part of the same directory name

    - Example: `CD Hello World`

- **Valid Arguments:**

    - .. (two period characters)
        - moves up to the parent directory.
        - Example: `CD ..`
    - absolute path
        - Navigates to specified path
        - Example: `CD /home/evan/Downloads`
    - relative path
        - Navigates to specified directory name in the current path
        - Example: `CD Downloads`

- **Response**:
    - The server sends back the updated working path directory as a string on success.
    - On failure, it sends back an appropriate error code
        - `DDNE`: Directory does not exist
        - `PD` : Permission Denied

- On success, the client updates the current directory indicator in the shell to match the new server directory.

- On error, the client displays the error code in easily understood format to the user. It does not update the shell path indicator.

- **Data Validation**
    - The client receives a string regardless of the outcome of an attempted `cd` command. Therefore, the previously mentioned method of data validation still works.

DOWNLOAD

- The server gets the filename request from the client and sends the given file to the client
- The download function was designed with larger files in mind. The buffer is able to transfer 1mb at a time.
- I have tested the download function with a 500 mb file and it worked great. It works with small text files as well.
- **Valid Arguments**
    - The server accepts 1 filename at a time. If the request has multiple words the server will interpret it as a single filename.
        - Example: `DOWNLOAD Homework 1.pdf`
- **Data Validation**
    1. Client sends download request.
    2. Server makes sure the file exists and is readable.
        - Server sends `READY` response to the client if successful
        - Server sends `FNF` (file not found) response if the file cannot be downloaded
    3. Client receives response and makes sure a file of the same name does not already exist.
        - If the file already exists, user is prompted to cancel, overwrite, or download the file under a different name.
    4. Client sends `READY` response to server if the user has chosen not to cancel
        - Otherwise, the client sends `STOP` and the server aborts download as well 5.** Server sends the length of the file it will send in bytes as a String with a return character ('\n') at the end**
    5. Client receives the size of the new file and saves it.
    6. The server and client write and read, respectively, from their output and input streams.
    7. The client keeps track of how many bytes it has read and stops when the number is equal to the file size received from the server.

        - This allows the client to be sure that it has read all data and has not lost any information over the network.

Issues

No issues reported.