FileTransfer: Design Document

Nicole Jingco
Tomas Quat
2020/10/06

Overview

To implement a simple client-server using the TCP/IP protocol Suite

Requirements

The following are the requirements for the application:

- Design and implement a server application that will listen for connections from clients and once the connection has been established it will respond to file transfer commands from the client(s)
- Design and implement a client application that will initiate a connection to a remote server, and will issue commands to the server to either send or receive a file
- Supply the server information as command line arguments
- Implement two functions: GET and SEND in the application
- Client can use the GET function to request a named file from the server
- Client can use the SEND function to send a named file to a remote server

Constraints

- You may use any language of your choice to implement your application
- The server will listen on port 7005 for connection requests
- When a GET or SEND command is received by the server initiate a data connection to send or receive a file
- Implementing a client/server architecture that utilizes a control channel for receiving/sending requests and a separate data channel to transfer files.

Design

The server was designed and implemented to support multiple concurrent clients.

Client

- Prompt user for data port #
- Connect to server listening port
- Send file transfer request (GET/SEND)
- Wait for confirmation
- Connect to server data port
- GET or SEND file data

Server

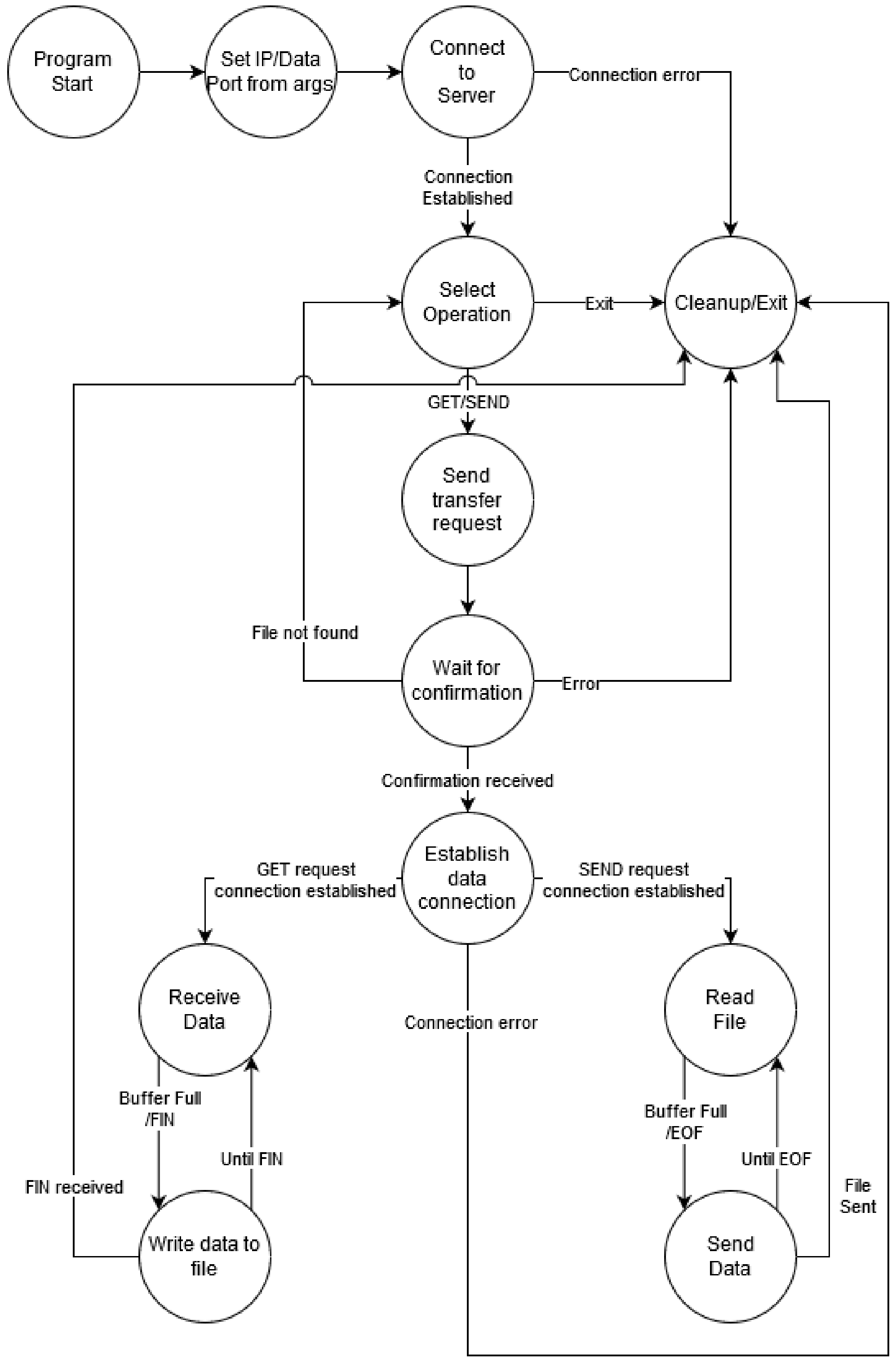
Main process:

- Listen for connections on port 7005
- Create child process to handle new client

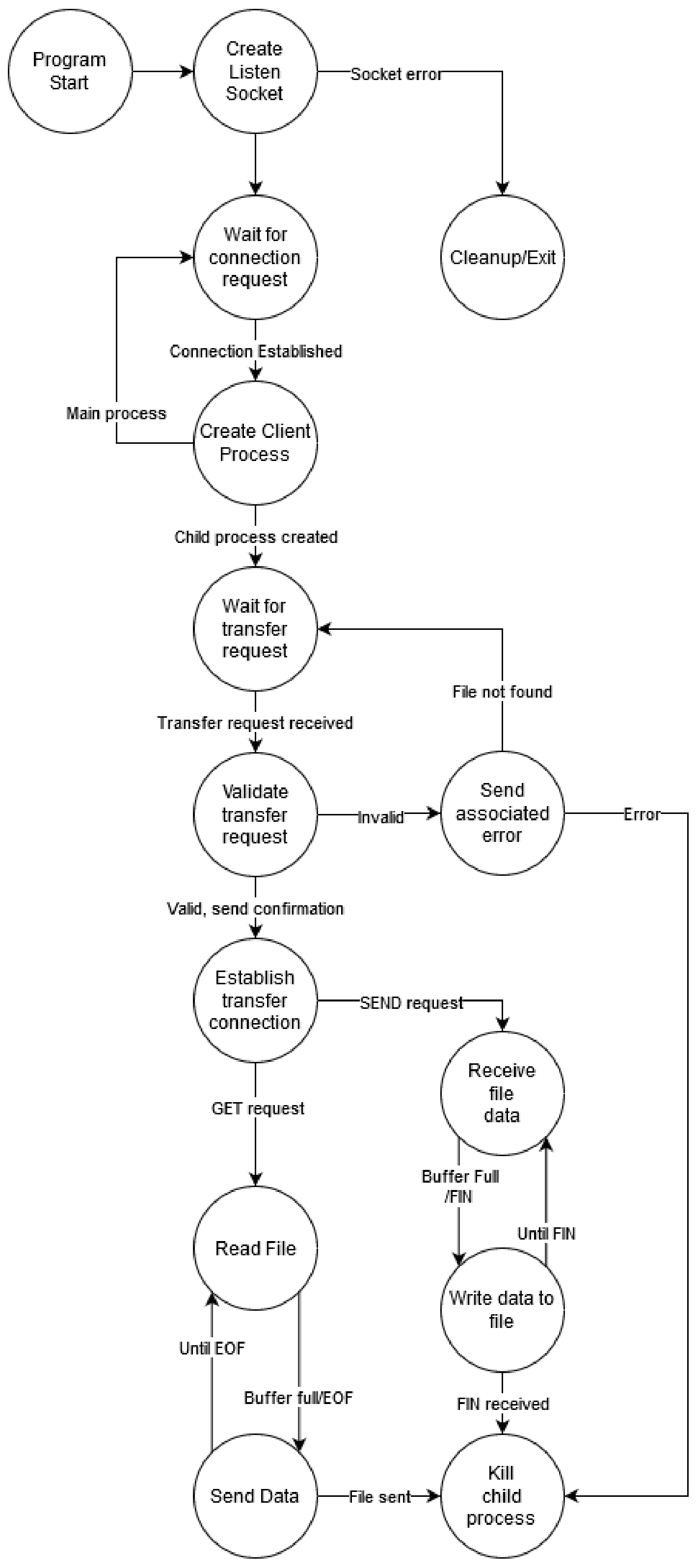
Child processes:

- Wait to receive transfer request
- Listen for data connection on provided port
- GET or SEND file data
- End child process

Client State Diagram



Server State Diagram



Pseudocode

Client

Set Ip/Data Port from args

```
Get IP and Port number from the user arguments
Go to ‘Connect to Server’
```

Connect to Server

```
Create socket
Attempt connection to server listen port
If successful:
    Go to ‘Select operation’
If error:
    Go to ‘Cleanup/Exit’
```

Select operation

```
Prompt user for operation and filename
If operation is Exit
    Go to ‘Cleanup/Exit’
If operation is GET or SEND
    Go to ‘Send transfer request’
If inputs are valid
    Create request packet
```

Send transfer request

```
Load transfer request, data port #, and filename into send buffer
Send request to Server
Go to Wait for confirmation
```

Wait for confirmation

```
If confirmation received:
    Go to ‘Establish data connection’
If file not found received:
    Go to ‘Select operation’
If error received:
    Go to ‘Cleanup/Exit’
```

Establish data connection

```
Create socket
Attempt connection to server data port
```

```
If successful && operation is GET
    Go to ‘Receive Data’’
If successful && operation is SEND
    Go to ‘Read File’’
```

Receive Data

```
Read received data into buffer
If buffer full or FIN received:
    Go to ‘Write data to file’
```

Write data to file

```
Create/Open requested file in /files/ directory
If FIN:
    Go to ‘Cleanup/Exit’
Otherwise:
    Write data to file
    Clear buffer
    Go to ‘Receive Data’
```

Read File

```
Read file data into buffer
If buffer full or EOF:
    Go to ‘Send Data’
If EOF sent
    Send FIN
```

Send Data

```
Send data to server
If EOF:
    Go to ‘Cleanup/Exit’
Otherwise:
    Clear buffer
    Go to ‘Read File’
```

Cleanup/Exit

```
Close all sockets
Exit program
```

Server

Create Listen Socket

```
Create and bind listen socket to port 7005
If successful:
    Go to ‘Wait for connection request’
If error:
    Print error
    Go to ‘Cleanup/Exit’
```

Wait for connection request

```
Loop/Block waiting for client connection
On connection:
    Go to ‘Wait for transfer request’
```

Create Client process

```
Create child process to handle the newly connected client
Main process:
    Go to ‘Wait for connection request’
New child process:
    Go to ‘Wait for transfer request’
```

Wait for transfer request

```
Wait for client transfer request to be received
On request received:
    Go to ‘Validate transfer request’
```

Validate transfer request

```
Parse transfer request into command, port, and filename
If command is not GET or SEND:
    Go to ‘Send associated error’
If port cannot be parsed:
    Go to ‘Send associated error’
If requested file does not exist:
    Go to ‘Send associated error’
Otherwise:
    Go to ‘Establish transfer connection’
```

Send associated error

```
If command is not GET or SEND:
    Send ERROR message to client
    Go to ‘Kill child process’
If port cannot be parsed:
    Send ERROR message to client
    Go to ‘Kill child process’
If command is GET && file does not exist:
    Send file not found message to client
    Go to ‘Wait for transfer request’
```

Establish transfer connection

```
Create and bind data socket to associated data port
Wait for client data connection request
On connection:
    Accept connection on specified port
If GET request:
    Go to ‘Read File’
If SEND request:
    Go to ‘Receive file data’
```

Read File

```
Read data from file until buffer full or End-of-File
Go to ‘Send Data’
```

Send Data

```
Send data to client
Clear data buffer
If not End-of-File:
    Go to ‘Read File’
If End-of-File:
    Go to ‘Kill Child process’
```

Receive file data

```
Read data into buffer
If buffer full or FIN:
    Go to ‘Write data to file’
```

Write data to file

```
Create/Open requested file in /files/ directory
If FIN:
    Go to ‘Kill child process’’
Otherwise:
    Write data to file
    Clear buffer
    Go to ‘Receive file Data’
```

Kill Child process

```
Close data socket
Close open files
Free allocated memory
Kill child process
```

Cleanup/Exit

```
Close sockets
Close open files
Free allocated memory
Exit program
```