Computer-Networks-FTP-Program-Tips

For more notes visit

https://rtpnotes.vercel.app

For lab programs visit

https://pgmsite.netlify.app

FTP-Client

Include Statements

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <sys/wait.h>
```

FTP-Client: Basic Algorithm for remembering

Main function

1. Create the socket
2. Setup the server address struct
3. Connect to the server
4. Loop
    1. Read get or close
    2. if the command is get
        1. Send get command to the server
        2. Execute get_function
    3. If the command is close
        1. Send close command to server
        2. Break the loop
5. Close the control socket

get_function

1. Enter filename
2. Send filename to the server
3. Receive validity check from server (Verifying whether the file exists or not)
4. Create data socket
5. Setup data connection address struct
6. Receive file data from server
7. Create a new file and write the file data
8. Close the file
9. Close the data socket

FTP-Client: Steps in more detail

Main function

1. Create the socket

```
csd = socket(AF_INET, SOCK_STREAM, 0);
```

2. Setup server address struct

```
servaddr.sin_family = AF_INET;
servaddr.sin_addr.s_addr = inet_addr("127.0.0.1");
servaddr.sin_port = 2000;`
```

3. Connect to the server

```
connect(csd, (struct sockaddr *)&servaddr, sizeof(servaddr))
```

4. Loop

4.1 Read get or close

```
scanf("%s", command);

if (strcmp(command, "get") == 0) {
        // Send 'get' command to the server
        send(csd, command, sizeof(command), 0);
        get_function(csd, data, argv[1]);
} else if (strcmp(command, "close") == 0) {
        // Send 'close' command to the server and break the loop
        send(csd, command, sizeof(command), 0);
        break;
} else {
        printf("Invalid command\n");
}
```

5. Close control socket and data socket

```
close(csd);
```

```c
close(dd);
```

get_function

1. Read filename and send filename to server

```c
printf("Enter filename: ");
scanf("%s", name);
printf("Name of file=%s\n", name);

// Send filename to server
send(csd, name, sizeof(name), 0);
```

2. Receive validity check

```c
recv(csd, valid, sizeof(valid), 0);
```

3. Create data socket and setup struct

```c
int dd = socket(AF_INET, SOCK_STREAM, 0);

data.sin_family = AF_INET;
data.sin_addr.s_addr = inet_addr(data_addr);
data.sin_port = htons(8080);
```

4. Connect to server and receive data

```c
connect(dd, (struct sockaddr *)&data, sizeof(data)) < 0)

recv(dd, r_Buf, sizeof(r_Buf), 0) < 0)
```

5. Write data to file

```
FILE *fp = fopen(name, "w");
        fprintf(fp, "%s", r_Buf);
        fprintf(fp, "FILE %s RECEIVED FROM SERVER WITH PROCESS ID = %d\n",
name, getpid());
        fclose(fp);
```

FTP-Server

FTP-Server: Basic Algorithm for remembering

Main function

1. Create the socket for control communication
2. Setup the control socket address struct
3. Bind the control socket
4. Listen for incoming connections
5. Accept connection from client
6. Loop
    1. Receive command from client
    2. If command is get
        1. Receive filename from client
        2. Open the requested file and send a success message
        3. Create a socket for data transfer
        4. Setup Data socket address structure
        5. Bind the data socket
        6. Listen for incoming connections
        7. Accept connection from client
        8. Execute send_file function
        9. send file content to the client
        10. Close connection, file and data socket
7. Close the connection and socket

send_file function

1. Read the file character by character and store it in buffer
2. Return the buffer

FTP-Server: Steps in more detail

Main function

1. Create socket for control communication

```
lfd = socket(AF_INET, SOCK_STREAM, 0);
```

2. Setup control socket address struct

```
control.sin_family = AF_INET;
control.sin_port = htons(port);
control.sin_addr.s_addr = INADDR_ANY;
```

3. Bind the control socket

```
bind(lfd, (struct sockaddr *)&control, sizeof(control)) < 0)
```

4. Listen for incoming connections

```
listen(lfd, 5);
```

5. Accept connection from client

```
socklen_t n = sizeof(client); confd = accept(lfd, (struct sockaddr
*)&client, &n);
```

6. Loop

6.1 Receive command from client

```
recv(confd, rBuf, sizeof(rBuf), 0);
```

6.2 Open the requested file and send success message to client

```
fp = fopen(rBuf, "r");
send(confd, "success", sizeof("success"), 0);
```

6.3 Create socket for data transfer and setup data structure

```
fd = socket(AF_INET, SOCK_STREAM, 0);

data.sin_family = AF_INET;
data.sin_addr.s_addr = INADDR_ANY;
data.sin_port = htons(8080)
```

6.4 Bind and listen for connections

```
bind(fd, (struct sockaddr *)&data, sizeof(data)
listen(fd, 5);
```

6.5 Accept connection and execute send_file function

```
n = sizeof(data_client);
confd2 = accept(fd, (struct sockaddr *)&data_client, &n);
send_file(fp);
```

6.6 Send file buffer to the client

```
send(confd2, send_buf, sizeof(send_buf), 0)
```

6.7 Close the file, connection and data socket

```
fclose(fp);
                close(confd2);
                close(fd);
```

7. Close the control socket and exit

```
close(confd);
close(lfd);
```

send_file function

```
void send_file(FILE *fp) {
    char ch;
    int i = 0;

    // Read the file character by character and store it in the buffer
    while ((ch = fgetc(fp)) != EOF) {
        send_buf[i++] = ch;
    }
    // Null-terminate the buffer
    send_buf[i] = '\0';
}
```