Concurrent FTP Server and Client

Aim: - Program to implement concurrent FTP server and client for file transfer to server.
Description -
Fork() creates a **new child process** that runs in sync with its **Parent process** and returns **0** if child process is created successfully.

- Whenever a new client attempts to connect to the TCP server, we will create a new **Child Process** that is going to run in parallel with other clients' execution.
  A **pid_t (Process id) data type** will be used to hold the Child's process id. Example: **pid_t = fork( )**.

This is the simplest technique for creating a concurrent server. Whenever a new client connects to the server, a fork() call is executed making a new child process for each new client.

- **Multi-Threading** achieves a concurrent server using a single processed program. Sharing of data/files with connections is usually slower with a **fork()** than with threads.

Steps involved in writing the Server Process:

1. Create a socket using socket( ) system call with address family AF_INET, type SOCK_STREAM and default protocol.
2. Initialize address structure with NULL, assign port number and IP address to the socket created.
3. Bind server's address and port using bind( ) system call by binding the socket id with the socket structure
4. Listen for active TCP connections (upto 10) in the socket file descriptor.
5. Wait for the client connection to complete accepting connections using accept( ) system call.
6. Display information of connected client and print the number of clients connected till now.
7. Create a new child process for each client using fork() system call.
8. Receive the Clients file using recv() system call .
9. Using *fgets(char *str, int n, FILE *stream) function, we read a line of text from the specified stream and stores it into the string pointed to by str. It stops when either (n-1) characters are read, or when the end-of-file is reached.
10. On successful execution i.e. when file pointer reaches end of file, file transfer "completed" message is sent by the server to the accepted client connection using newsd, socket file descriptor.

Steps involved in writing the Client Process:

1. Create a socket system call with address family AF_INET, type SOCK_STREAM and default protocol.
2. Initialize address structure with NULL, assign port number and IP address to the socket created.
3. Enter the client port id
4. Connect to the server address using connect() system call.
5. Read the existing and new file name from user.
6. Send existing file to server using send() system call
7. Receive feedback from server "Completed", regarding file transfer completion.
8. Display the message in the file on the clients screen.
9. Write "File is transferred" message to standard output screen of client and exit.
10. Close the socket connection.

Program – ftpserver.c

```
// Server side program that accepts connection from
// every client concurrently

#include <arpa/inet.h>
#include <netinet/in.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <unistd.h>

// PORT number
#define PORT 4444

int main()
{
    // Server socket id
    int sockfd, ret,n;
    char rcv[100],fileread[100];
```

```
// Server socket address structures

FILE *fp;
struct sockaddr_in serverAddr;

// Client socket id
int clientSocket;

// Client socket address structures
struct sockaddr_in cliAddr;

// Stores byte size of server socket address
socklen_t addr_size;

// Child process id
pid_t childpid;

// Creates a TCP socket id from IPV4 family
sockfd = socket(AF_INET, SOCK_STREAM, 0);

// Error handling if socket id is not valid
if (sockfd < 0) {
    printf("Error in connection.\n");
```

```
    exit(1);
}

printf("Server Socket is created.\n");

// Initializing address structure with NULL
memset(&serverAddr, '\0',
       sizeof(serverAddr));

// Assign port number and IP address
// to the socket created
serverAddr.sin_family = AF_INET;
serverAddr.sin_port = htons(PORT);

// 127.0.0.1 is a loopback address
serverAddr.sin_addr.s_addr
    = inet_addr("127.0.0.1");

// Binding the socket id with
// the socket structure
ret = bind(sockfd,
           (struct sockaddr*)&serverAddr,
           sizeof(serverAddr));
```

```
// Error handling
if (ret < 0) {
    printf("Error in binding.\n");
    exit(1);
}

// Listening for connections (upto 10)
if (listen(sockfd, 10) == 0) {
    printf("Listening...\n\n");
}

int cnt = 0;
while (1) {

    // Accept clients and
    // store their information in cliAddr
    clientSocket = accept(
        sockfd, (struct sockaddr*)&cliAddr,
        &addr_size);

    // Error handling
    if (clientSocket < 0) {
```

```
            exit(1);
        }

        // Displaying information of
        // connected client
        printf("Connection accepted from %s:%d\n",
               inet_ntoa(cliAddr.sin_addr),
               ntohs(cliAddr.sin_port));

        // Print number of clients
        // connected till now
        printf("Clients connected: %d\n\n",
               ++cnt);


        // Creates a child process
        if ((childpid = fork()) == 0) {
 n=recv(clientSocket,rcv,100,0);
rcv[n]='\0';
fp=fopen(rcv,"r");
if(fp==NULL)
{
send(clientSocket,"error",5,0);
```

```
close(clientSocket);
}
else
{
while(fgets(fileread,sizeof(fileread),fp))
{
if(send(clientSocket,fileread,sizeof(fileread),0)<0)
{
printf("Can't send file contents\n");
}
sleep(1);
}
if(!fgets(fileread,sizeof(fileread),fp))
{
//when file pointer reaches end of file, file transfer "completed" message is send to accepted client connection using newsd, socket file descriptor.
send(clientSocket,"completed",999999999,0);
}
}
            // Closing the server socket id
            close(sockfd);

        }
```

```
    }

    // Close the client socket id
    close(clientSocket);
    return 0;
}
```

ftpclient.c

```
// Client Side program to test
// the TCP server that returns
// message in a file

#include <arpa/inet.h>
#include <netinet/in.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <unistd.h>

// PORT number
#define PORT 4444
```

```
int main()
{
    // Socket id
    int clientSocket, ret,s;
    FILE *fp;
    // Client socket structure
    struct sockaddr_in cliAddr;

    struct sockaddr_in serverAddr;

    // char array to store incoming message
    char buffer[1024],name[100],fname[100],rcvg[100];

    // Creating socket id
    clientSocket = socket(AF_INET,
                          SOCK_STREAM, 0);

    if (clientSocket < 0) {
        printf("Error in connection.\n");
        exit(1);
    }
    printf("Client Socket is created.\n");
```

```
// Initializing socket structure with NULL
memset(&cliAddr, '\0', sizeof(cliAddr));

// Initializing buffer array with NULL
memset(buffer, '\0', sizeof(buffer));

// Assigning port number and IP address
serverAddr.sin_family = AF_INET;
serverAddr.sin_port = htons(PORT);

// 127.0.0.1 is Loopback IP
serverAddr.sin_addr.s_addr
    = inet_addr("127.0.0.1");

// connect() to connect to the server
ret = connect(clientSocket,
              (struct sockaddr*)&serverAddr,
              sizeof(serverAddr));

if (ret < 0) {
    printf("Error in connection.\n");
    exit(1);
}
```

```
    printf("Connected to Server.\n");


    while (1) {
 printf("Enter the existing file name\t");
scanf("%s",name);
printf("Enter the new file name\t");
scanf("%s",fname);
fp=fopen(fname,"w");
while(1)
{
send(clientSocket,name,sizeof(name),0);
s=recv(clientSocket,rcvg,100,0);
if(s<0)
printf("Error in receiving data");
else
{
rcvg[s]='\0';
if(strcmp(rcvg,"error")==0)
{
printf("File is not available\n");
}
if(strcmp(rcvg,"completed")==0)
```

```
{
printf("File is transferred.....\n");
break;
fclose(fp);
close(clientSocket);
}
else
{
// Printing the message on screen
fputs(rcvg,stdout);
fprintf(fp,"%s",rcvg);
bzero(rcvg,sizeof(rcvg));
}
}
}
return 0;
} }
labb30@labb30:~$ gcc server.c -o s
labb30@labb30:~$ ./s
Server Socket is created.
Listening...

Connection accepted from 0.0.0.0:0
```

Clients connected: 1

Connection accepted from 127.0.0.1:48308

Clients connected: 2

Connection accepted from 127.0.0.1:48310

Clients connected: 3

Connection accepted from 127.0.0.1:48312

Clients connected: 4

gcc client.c -o c1

labb30@labb30:~$ ./c1

Client Socket is created.

Connected to Server.

Enter the existing file name hello.txt

Enter the new file name hello1.txt

welcome to lmcst

File is transferred.....

gcc client.c -o c2

labb30@labb30:~$ ./c2

Client Socket is created.

Connected to Server.

Enter the existing file name dc.txt

Enter the new file name dc1.txt

Have a nice day!

File is transferred.....

gcc client.c -o c3

labb30@labb30:~$ ./c3

Client Socket is created.

Connected to Server.

Enter the existing file name hi.txt

Enter the new file name hi1.txt

Best Wishes!

File is transferred.....

gcc client.c -o c4

labb30@labb30:~$ ./c4

Client Socket is created.

Connected to Server.

Enter the existing file name gh.txt

Enter the new file name gh1.txt

Best of Luck.

File is transferred.....