A File Transfer Protocol: Concurrent Server Design

CS4254: Project 2
Fall 2004
Deadline: 11/11/2004 (by midnight)

Introduction

In the first project, you implemented a simple file transfer protocol between a client and a server. However, your implementation had a significant limitation. At any given time, the server could only handle one file transfer client. In this project, you need to augment you server program to handle multiple clients.

For the purposes of this project, you will implement three different mechanisms that are commonly used to handle concurrency at the server. They are:

- True concurrency using the fork system call. This is the simplest form of concurrency from an implementation standpoint. When a client initiates a connection to the server, the server uses the fork call to create a new copy of the server process, which then handles the client request.
- Concurrency based on threads. In this form of concurrency, when a client initiates a connection, the server creates a new thread to service the client.
- Apparent concurrency based on asynchronous IO. In this form of concurrency, the server uses asynchronous IO calls to switch between servicing different clients.

Implementation

The implementation of this project consists of two parts. In the first part, you will implement the server side of your project in C as three independent programs, each of which implements an ftp server based on one of the concurrency mechanisms listed above. The second part of the project consists of modifications to the client to allow it to work in a multi-client environment. The specifications for the ftp_client and ftp_server programs are similar to the first project.

To ensure that the ftp_client program can work in a multi-client environment, you need to ensure that the data port on the different client programs runs on different port addresses. Recall that the ftp_client program acts as the server for the data port. Obviously, the current scheme of using the last 4 digits of your SSN + 1 for the data port address will not work. Why? (Hint: What happens if two ftp_client programs running on the same machine try to connect to different ftp servers?).

To make the ftp_client program safe for a multi-client environment, each client should run the dataport on a different port address. You can get a unique port address on the client side by using a wild card for the port number. The default wildcard is a value of 0 for the port number. If the data port server uses a port number of 0, the OS automatically assigns a unique port to it after the bind call. You can use this mechanism to ensure that the ftp_client always uses a unique port for the server end of the data port connection.

However, there are two problems. First, in order for the ftp_server to be able to connect to the data port, you need to determine the port number assigned to the server end of the data port at the ftp_client. This can be accomplished by calling the getsockname() function after binding to the server socket in the ftp_client program. Take a look at the man page for the getsockname() function. The next problem is that you need to communicate the port address of the data port to the server. This can be accomplished by sending the port number with each command that requires the use of the data port (e.g. list, put, get).

In this architecture, the ftp_client program first creates the server port, binds to it and retrieves the server port number with getsockname(). It then sends the command typed by the user to the server over the control port and adds the data port number to the end of the command. After sending the command to the server, the client initiates the blocking accept call on the data port. When the server receives this information, it can use the data port number to connect back to the client. Since different instantiations of the client program will be assigned different port numbers by the OS, your client will be safe to use in a multi-tasking environment.

Notes for the FTP server implementation:

1. For the threads implementation, you need to use the POSIX compatible **pthreads** library on Linux. **No other OS platform or threads library will be accepted.**
2. Test each of your server implementations thoroughly by running multiple clients against the server.

Submission information

NOTE: For this project up to 2 students may pair up to form a group.
Your code should be implemented as C programs that run under Linux. You need to submit an electronic version of your project to chekhov.cs.vt.edu by logging in under 1 member of the group's id number.

The submission should contain

- Source code
- All binaries

Your source code should also contain a comment on the first few lines with the authors of the code and the VT ID numbers of the authors.

The submission format is tar. Use the tar program to archive the contents of your project directory as follows:
If your project directory is called `project2` and it is under your home directory, then go to your home directory and issue the following command:

```
tar cvf filename.tar project2
```

`filename.tar` now contains the archived version of your project directory.