CS342 Operating Systems, Spring 2012

Project 1: Processes, Threads, and Communication

Assigned: 01.03.2011
Due date: 15.03.2011

> Tell me and I'll forget;
> Show me and I may remember;
> Involve me (I do) and I'll understand (I learn).
> *Chinese proverb*

In this project, you will implement a file download application. You will implement both the server side and the client side. Hence you will implement two programs, a file server program and a file client program. They can run on the same machine or on different machines.

The file server program will be called as `fileserver`. It will be invoked as follows:

fileserver <port>

When started, it will become a process which we will call as the main server process in this document. The main server will wait for clients to connect to it using the indicated port number. The IP address (and machine name) used by the server will be the IP address (and the machine name) where the server process is started.

The file client program will be called `fileclient` and it will be invoked as follows:

fileclient <serverip or servername> <serverport>

When started, a fileclient process will connect to the server. Upon accepting the connection request from the client process, the main server process will create a server process (a child) and all communication (data and control message exchange) from/to client will be done by the newly created server process.

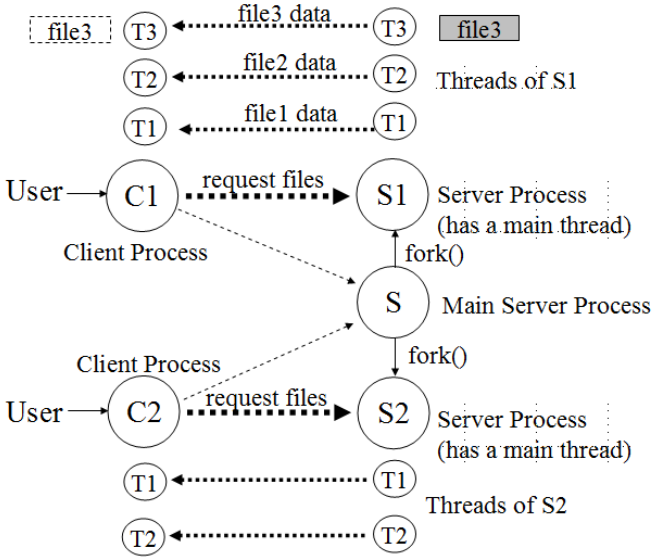
When started, the fileclient process will provide the user a prompt like ">". At the prompt, a user can type two commands: 1) list; 2) get; 3) quit. The list command does not take any parameters. The get command may take one or more filenames (upto 5) as parameters. When list command is sent to the server, the server will send back the names of the files available in its directory to be downloaded. When get command is sent with the names of some files, the server will send those files to the client. For each filename specified, the client and server proceses will be creating a separate thread to download/upload the specified file. For example, if there are 3 files to download, 3 threads (tc1, tc2, tc3) at the client side and 3 threads (ts1, ts2, ts3) at the server side (i.e. server child process side) will be created. File 1 will be uploaded by ts1 to tc1 and file 2 will be uploaded by ts2 to tc1, and so on. The client process will wait until all created threads terminate, and after that it will again type a prompty (">") to the screen so that the user can request another set of files. The corresponding server process will wait until all created threads finish their job and terminate. Then the (child) server process can wait for another request from the respective client.

When the user writes quit command at the prompt of a fileclient, the fileclient process will terminate. The corresponding server process should be terminated at that time as well (it is upto you to design the protocol).

The protocol between the client and server to request and download files will be designed by you. It is up to you which messages to use and in which sequence, etc. If you wish you can use a control connection between a client and server (to exchange control messages, not file data messages).

There can be more than one fileclient connected to the server at the same time. For each such file client, the main server creates and has a separate (child) server.

The figure below explains the run-time structure of the application.



As we see, the main server process is always up and running. A server process (i.e. a child process) is only created when there is a client process to get service. A server process will be alive as long as the corresponding client process is alive.

A downloaded file will be written to the disk of the fileclient. If a requested file is not available at the server, the server should indicate this properly to the client so that the client will not wait forever for file-data to arrive.

Below are examples for the invocations of the server and clients. Assume the machine mypc.cs.bilkent.edu.tr has an IP address 139.179.35.46.

./fileserver 8915

./fileclient mypc.cs.bilkent.edu.tr 8915
./fileclient 139.179.35.46 8915
./fileclient 139.179.35.46 8915

Below are examples of what we can type the prompt of a file client:

>list
>get file1.txt file45.pdf fx.bin
>get file3.gif
>get
>quit

Submission

You will submit online. You will submit both your programs and a Makefile, so that we can just type make and obtain the executable files: fileserver and fileclient. Do not include binary (executable) files in your upload.

Notes:

- Max filename size can be 255 characters.
- We will specify a filename, not a pathname.
- The server will serve the files sitting in the directory where it is started.
- You can use the getaddrinfo() function to learn the IP address of a machine whose name is known. Read the related manual page (man page).
- You will use TCP sockets.

Suggested References:

1. Unix Network Programming, Volume 1, W. Richard Stevens, Networking APIs: Sockets and XTI, Second Edition, Prentice Hall PTR, 1998.
2. Unix Network Programming, Volume 2, W. Richard Stevens, Interprocess Communications, Second Edition, Prentice Hall PTR, 1998.