Lab 5. Streaming Fileserver

Introduction

The last lab (Lab 4. Stateless File Server and Client) introduced us to a simple fileserver using UDP transport. This time around we will be exploring the other transport protocol, TCP.

Outcomes Addressed

- understand what networking protocols are and how they are specified
- understand the protocols of the internet
- be able to write applications using socket connections
- understand the implementation and operation of Internet services

Background

The Lab 4. Stateless File Server and Client was designed to be minimally complex, and would likely be used by a firmware bootloader that does not have a full TCP/IP stack available. UDP is a thin API over IP datagrams, and even the IP protocol could be "hardcoded" into the Ethernet frame for a minimal implementation. The approach has some other benefits, but performance is not going to be one of them.

While TCP will have additional overhead associated with establishing the connection, it should afford much higher performance due to the sliding window approach to transporting packets. Also, we will be able to request a file with a single request instead of multiple requests.

Lab Activity

Basics

If it not obvious by now, the goal of this lab will be to write a streaming file server and client combo. Actually, half of the class will write a server, and half will write the client, and all of them shall be able to inter-operate.

This technically will not be a stateful fileserver, as each transaction will be completed in one step. An example of a stateful file server might be ftp, which required login/authentication, multiple file downloads per session, etc.

Although the specification could be changed dramatically from the previous lab, let's keep as much the same as possible.

- Server Properties
  - Listens on port 22222, but port can be modified via command-line argument
  - All communication will be to same port
  - Will only serve files located in a particular directory / directory subtree
  - Will only serve files, will not allow upload
  - No security / passwords / authentication features

    - Will serve up a directory listing as a text file if file "." is requested
    - Will respond to whatever port client used (as src port)
    - Server disconnects upon sending last byte of requested file, which will transmit an EOF to the client

- Client Properties
    - Simple command-line interface (or GUI if you choose)
    - Defaults to contacting server at port 22222, but port can be changed via command-line option
    - Will make a single request for a file, and expect a single response with the entire streamed file
    - Stores downloaded file in current directory or some default directory specified by option

The server is stateless, so, as mentioned above, if it receives a valid request, it will comply. If the request is not valid, it will response with an appropriate error code. Any given request should not depend on any previous request.

You are welcome to implement in any language on any platform, but your instructor can only support Sockets on Linux in 'C'. Most other APIs are based on Unix Sockets. You should be using an upper-level API, and not something like WinPCap.

Protocol

This streaming version will use the same request/reply packet format, although a client will essentially only make one request for a file, and the entire file will be streamed to the client after the response packet. Review the information below, with the necessary changes in **bold**.

The SSFTP (Super Stateless File Transfer Protocol) is as follows. The base packet format is identical for both the request and response.

<table>
<tr><th>byte 0</th><th>byte 1</th><th>byte 2</th><th>byte 3</th></tr>
<tr><td>Flags</td><td colspan="2">Length</td><td>Offset</td></tr>
<tr><td colspan="3">(cont)</td><td>File-</td></tr>
<tr><td colspan="4">name</td></tr>
<tr><td colspan="4">(cont)</td></tr>
<tr><td colspan="4">(cont)</td></tr>
<tr><td colspan="4">(cont)</td></tr>
<tr><td colspan="4">(cont)</td></tr>
<tr><td colspan="4">(cont)</td></tr>
<tr><td colspan="4">(cont)</td></tr>
<tr><td colspan="3">(cont)</td><td>Data</td></tr>
<tr><td colspan="4">(cont)</td></tr>
</table>

Field Descriptions

- Flags (bit 0 is LSB)

    - bit 0 : 1 = request, 0 = response
    - bit 1 : 1 = eof encountered (response)
    - bit 2 : 1 = file not found (response)
    - bit 3 : 1 = invalid request (response)
    - bits 4..7 : not used, should be 0
- Length: 16-bit unsigned integer length of requested file block (request), length of data field returned (response). **For this version, length will be the suggested size of each chunk of the file to be sent at a time (client request to server). Server response can set this field to 0.**
- Offset: 32-bit unsigned integer offset within a requested file to start supplying data (request and response). **For this version, this should always be 0.**
- Filename: 32 byte ASCII character array with requested filename. Filename will be left-justified, and null-terminated.
- Data: not included in request, contains requested file data in response.

Note: be sure to read and write file with binary file i/o.

The server's reply will mirror back the starting offset **(0)** and filename so the client can properly identify the response.

If the client has requested a file listing (file name ”.”, no quotes), the server will respond with a list of files available. The lists will be transmitted similar to a text file with a linefeed character as a delimiter ('\n') between each filename. The last filename in the list will need the delimiter as well.

The packet will be filled in network order.

Additional Requirements

In concert with your client/server buddy, run some benchmarks on the performance of this server versus the Lab 4. Stateless File Server and Client. Be sure to include a sufficient number of trials and file sizes to draw a conclusion. Include benchmark results in tabular form in your pdf report.

Resources

- TCP Echo Server shows a simple echo server, that uses similar messaging techniques required for this project.

Deliverables

This is a two-week lab

1. Demonstrate a working client or server during lab in Week 8 (due at beginning of lab).
2. Submit all source code in a single pdf file and benchmark results. Submit via link on instructor's syllabus. Include in a comment block at the top of the source a paragraph relating your experience with this project and suggestions for improvement. Include any glaring deficiencies of your approach. Also include full user instructions.
3. Submit a zip file with all source code and support files. The submitted source should be buildable.
    a. Do not include pdf in zip file.
4. Pdf and zip submission is due prior to lab in Week 8. Pdf and zip must be submitted prior to demonstration.

5. All source code must be properly documented for full-credit.

- courses/ce-4960/labs/streaming_file_server.txt · Last modified: 2012/04/16 16:31 by rothede