Lab 4. Stateless File Server and Client

Introduction

Ever downloaded a file? Ever had that download interrupted? Ever had to restart that interrupted download from scratch? The interruptions may have occurred due to dropped packets which may have resulted in loss of state, that is, the file transfer protocol became confused and aborted the transfer. A stateless file server does not retain state of any active download(s). Rather, each request for a file is self-contained, as is each response. Of course, the ability to re-start a partial download lies in the intelligence of the client.

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

As mentioned above, stateless file servers are very simplistic, and do not keep track of sessions or connections or active downloads. Each request will contain a filename to download, the offset within that file, and the [max] length to send. The server will open the file, get the requested portion of the file, close the file, and send the requested information. Stateless file servers are inherently slooow for large files, so they are not widely used. However, the simplicity of the interaction makes them useful for transfers to and from machines that may not have a full TCP/IP stack, such as an embedded device downloading firmware.

While it is the responsibility of the transport layer protocols to provide reliability and connection support, user datagram protocol (UDP) actually provides none of this. UDP is a rather thin wrapper for IP datagrams and thus provides the ability to send and receive simple messages with a bit less complexity than is required when using the more full-featured transport protocol TCP. Since UDP does not provide reliability, it is necessary for the application providing its own reliability checks, usually through the use of acknowledgements, message identification, etc. UDP is ideal for this file server as the protocol is so simple, very few checks will need to be added at the application level.

Lab Activity

Basics

If it not obvious by now, the goal of this lab will be to write a stateless 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.

- 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. The filenames within the text file will be separated by newlines - '/n'.
    - Will respond to whatever port client used (as src port)
- 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
    - Defaults to requesting up 1400 bytes of the file at a time, but this can be changed via option up to max size of a UDP packet (65507 bytes)
    - 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 respond with an appropriate error code. Any given request should not depend on any previous request.

The client will need to keep track of requests, and be prepared to repeats requests if there is no reply from the server, or an error is indicated. Although the client does not need to make requests in-order, it does not really make sense to do it otherwise.

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

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

byte 0	**byte 1**	**byte 2**	**byte 3**
Flags	Length		Offset
(cont)			File-
name			
(cont)			
(cont)			
(cont)			
(cont)			
(cont)			
(cont)			
(cont)			Data
(cont)			

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).
- Offset: 32-bit unsigned integer offset within a requested file to start supplying data (request and response).
- 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.

Note: if a 'C' struct is used to represent the packet above, be aware of padding that may be added by the compiler. To prevent padding, add __attribute[1)] to struct declaration.

Resources

- UDP-based 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 6 (due at beginning of lab).
2. Submit all source code in a single pdf file. 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 6. Pdf and zip must be submitted prior to demonstration.
5. All source code must be properly documented for full-credit.

[1)] packed

- courses/ce-4960/labs/stateless_file_server.txt · Last modified: 2012/04/03 09:30 by rothede