Redes de Computadores

1st Semester 2014/2015
LEIC Alameda

Programming using the *Sockets* interface

"Content Delivery Network"

"A content delivery network (CDN) is an interconnected system of computers on the Internet that provides Web content rapidly to numerous users by duplicating the content on multiple servers and directing the content to users based on proximity."

1. Introduction

The goal is to develop an application for content distribution, with several servers replicating the available contents, with the user requests being answered by the "nearest" server.
We consider one *Central Server* (CS), which will be the users' contact point both to ask about the available contents as well as to make new contents available in the system. It is assumed that the user knows the URL where the central sever is available.

The interaction with the CS allows the user to get a list of the available contents and to get the identification of the *Storage Server* (SS) more suitable to retrieve contents from. The user application can then solicit the transfer of a given content from that SS server to a local directory.
The system also allows to upload new contents to the system, by contact with the central server, which then replicates the contents to all the available SS servers.

For the implementation, the application layer protocols operate according to the client-server paradigm, using the transport layer services made available by the socket interface, using the TCP and UDP protocols.

The central server accepts content list requests using the UDP protocol, identifies the SS server to be used for retrieving files, using the TCP protocol, by a given user. The central server accepts new contents via a TCP connection established by the user, and then replicates those contents to the SS servers using TCP connections.

2. Project Specification

2.1 User

The program implementing the client should be invoked using the command:

`./user [-n CSname] [-p CSport]`,

where:

`CSname` is the name of the é machine where the central server (CS) runs. This is an optional argument. If this argument is omitted, the CS should be running on the same machine.

`CSport` is the well-known port where the CS server accepts user requests, both in TCP and in UDP. This is an optional argument. If omitted, it assumes the value 58000+NG, where NG is the number of the group.

Once the user program is running, it waits for the user to indicate the action to take, notably:

- `list` – following this instruction the application should contact the CS, using the UDP protocol, asking the list of contents available in the SS servers for download. In its reply the CS also identifies the IP address and the port number of the SS server that the user should contact to download contents.
- `retrieve` *content_name* – the client application communicates in TCP with the SS (previously indicated by the CS in reply to the `list` command), to transfer the file *content_name* to the local directory and notifies the user that the transfer has been completed.
- `upload` *content_name* – the client application communicates in TCP with the CS to transfer the file *content_name* which is in the local directory to the CS. The CS should then synchronize with all existing SS servers and then update the list of contents available for download.
- `exit` – the client application terminates.

2.2 *Central Server* (CS)

The program implementing the *Central Server* should be invoked using the command:

`./CS [-p CSport]`,

where:

`CSport` is the well-known port where the CS server accepts user requests, both in TCP and in UDP. This is an optional argument. If omitted, it assumes the value 58000+NG, where NG is the number of the group.

The central server makes available two servers ,both with well-known port `CSport`, one supported in UDP to answer user listing requests, the other in TCP to receive new contents.

The CS server supported in UDP can answer the *LST* messages received from a user. The CS returns the list of contents available, as well as the IP and port numbers of the SS server that this user should then contact to retrieve the desired contents in TCP (after the user has given the *retrieve* instruction). The choice of the SS server can be done consulting a previously built table, or in alternative making a random selection among the available SSs.
The CS server supported in TCP answers user instructions of the type *upload* in which the client application transfers, in TCP, a new file to the CS, which should replicate it to all the SS servers and update its available contents list.
The CS server has a file listing the IP addresses and TCP port numbers for the active SS servers.

The CS servers output to the screen the received requests and the IP and port which originated those requests.
Each user request should be responded immediately.

2.3 *Storage Server* (SS)

The program implementing the *Storage Server* should be invoked using the command:

`SS [-p SSport]`,

where

SSport is the well-known port where the SS server accepts requests from the users. This is an optional argument. If omitted, it assumes the value 59000.

The SS server is supported in the TCP protocol with well-known port *SSport*.
The SS accepts user requests for the transfer of a given file. The SS outputs to the screen the received requests and the IP and port which originated those requests.

This server can also accept files sent by the central server. The SS outputs to the screen the received file name and that its origin is the CS.

3. Communication Protocols Specification

3.1 *User–CS* Protocol in UDP

The user program, following the `list` instruction interacts with the CS in UDP according to the following requests and replies:

a) `LST`
Following the `list` instruction, the user requests from the CS a list of the contents available in the SS servers.

b) `AWL IPSS portSS` n_T *F1 F2 … Fn*$_T$
In reply to a `LST` request the CS server replies in UDP indicating the IP address (`IPSS`) and the TCP port number (`portSS`) where the *Storage Server* (SS) that should be contacted is available, the number (n_T) and the list of files available (`F1 F2 …` Fn_T), where `Fn` is the name of file number `n`. The reply is sent by the CS immediately after receiving the request.
If the `LST` request cannot be answered (e.g., empty file) the reply will be `EOF`. If the `LST` request is not correctly formulated the reply is: `ERR`.

To simplify, you can assume that n_T is at most 30, and that each file name (`Fn`) contains no more than 20 characters.

Each message of request or reply ends with the character "`\n`".

3.2 *User–CS* Protocol in TCP

For the user to upload a new file to the system (following the `upload` instruction) a connection with the central server is established in TCP. It should be verified if the file already exists in the server, if not the *upload* will be allowed.
The protocol includes the following requests and replies:

a) `UPR` *Fn*
The user requests the CS to check whether a file with name *Fn* already exists in the server.

b) `AWR` *status*
Following a `UPR` request the response will be: `dup` in the `status` field if a file with the same name already exists, or with `new` in the `status` field if it is a new file. If the request is not well formulated the answer will be `ERR`.

c) `UPC` *size data*
The user sends the previously identified file in the `UPR` request, indicates its `size` in Bytes, followed by the corresponding data.

d) `AWC` *status*
Following a `UPC` request the CS response will be `ok` in the campo `status` field if the file has been successfully uploaded to the SS servers, or `nok` otherwise. If there is a protocol (syntax) error the answer will be `ERR`.

Each message of request or reply ends with the character "\n".

3.3 *User – SS server* Protocol

When the user wants to download a file, following the `retrieve` instruction, the application starts a TCP connection with the SS server previously identified (after consulting the CS following the `list` instruction).
The protocol includes the following requests and replies:

a) `REQ` *`Fn`*
The user requests the file with name *`Fn`* to the SS.

b) `REP` *`status size data`*
In reply to a `REQ` request the SS server replies with `ok` in the `status` field if it can send the file, and `nok` otherwise (e.g., file not found), followed by the dimension (`size`) of the file in Bytes and the data itself (*`data`*). If there is a protocol (syntax) error the answer will be `ERR`.

Each message of request or reply ends with the character "\n".

3.4 *CS –SS* Protocol

After a new file is uploaded to the CS server (following the `upload` instruction), the CS server should start a TCP connection with each of the SS active servers to send a copy of the file.
The protocol includes the following requests and replies:

a) `UPS` *`Fn size data`*

The CS server sends the file named *`Fn`*, with `size` Bytes, followed by the corresponding data.

b) `AWS` *`status`*

In reply to a `UPS` request the SS server replies with `ok` in the `status` field if it was able to successfully save the file, or `nok` otherwise. If there is a protocol (syntax) error the answer will be `ERR`.

Each message of request or reply ends with the character "\n".

4. Development

4.1 Development and test environment

Make sure your code compiles and executes correctly in the development environment available in lab LT5.

4.2 Programming

The operation of your program should be based on the following set of system calls:

- Computer name: `gethostname()`.
- Remote computer IP address from its name: `gethostbyname()`.
- UDP server management: `socket()`, `bind()`, `close()`.
- UDP client management: `socket()`, `close()`.
- UDP communication: `sendto()`, `recvfrom()`.
- TCP client management: `socket()`, `connect()`, `close()`.
- TCP server management: `socket()`, `bind()`, `listen()`, `accept()`, `close()`.
- TCP communication: `write()`, `read()`.
- Concurrent servers: `fork()`.

4.3 Implementation notes

Developed code should be adequately structured and commented.
The `read()` and `write()` system calls may read and write, respectively, a smaller number of bytes than solicited – you need to ensure that your implementation still works correctly.
Both the client and server processes should terminate gracefully at least in the following failure situations:

- wrong protocol messages received from the corresponding peer entity;
- error conditions from the system calls.

5 Bibliography

- W. Richard Stevens, Unix Network Programming: Networking APIs: Sockets and XTI (Volume 1), 2nd edition, Prentice-Hall PTR, 1998, ISBN 0-13-490012-X, chap. 5.
- D. E. Comer, Computer Networks and Internets, 2nd edition, Prentice Hall, Inc, 1999, ISBN 0-13-084222-2, chap. 24.
- Michael J. Donahoo, Kenneth L. Calvert, TCP/IP Sockets in C: Practical Guide for Programmers, Morgan Kaufmann, ISBN 1558608265, 2000
- On-line manual, `man` command
- Code Complete - http://www.cc2e.com/
- http://developerweb.net/viewforum.php?id=70

6 Project Submission

6.1 Code

The project submission should include the source code of the programs implementing the *user*, the *CS server* and the *SS server*, as well as the corresponding *Makefile*.

6.2 Submission

The project submission is done by e-mail to the lab teacher, **no later than October 10, 2014, at 8 PM**.
You should create a single `zip` archive containing all the source code and other files required for executing the applications. The archive should be prepared to be opened to the current directory and compiled with the command `make`. The name of the archive should follow the format: `proj< group number>.zip`

6 Questions

You are encouraged to ask your questions to the teachers in the scheduled foreseen for that effect.