LTH

EDA031 PROJECT

# News System

Andreas Fransson (atf09afr) 123.fransson@gmail.com
Harald Nordgren (ada09hno) ada09hno@student.lu.se

April 11, 2015

## Contents

## Introduction

This report assumes that the reader is familiar with the EDA031 (2015) project description and handouts[1]. To summarise, the report describes the implementation of a rudimentary network based news system, where the user can create/delete/access newsgroups and articles. The database comes in two versions, one memory based and one disk based, both are implemented in such a way that they are interchangeable from the perspective of the server which uses them, save for the fact that the memory one is empty when it starts. The server permits connections from multiple clients. The client itself includes a rudimentary text based interface which reflects the possible actions permitted by the database and pre-defined communication protocol.

[1]To save everyone's time.

## Component list

Before diving into how everything comes together (what uses what), it seems prudent to simply list everything with short explanations:

**server_main**
: Extension of the original myserver.cc. Contains a database object and functions which receive and respond to the different possible communication codes defined in the project description pdf.

**client_main**
: Extension of the original myclient.cc. Added are a few functions which constitute a terminal, as well as a set of functions which bridge the terminal with database server, according to the communication codes defined in the project description pdf.

**database (interface)**
: Abstract class for the two database types. All the functions in the header are declared as pure virtual functions. The functions correspond to the 7 protocol-defined operations: {list_newsgroups(); list_articles(); create_newsgroup(); create_article(); delete_newsgroup(); delete_article(); read_article()}.

  The return types are minimalistic, consisting of an *unsigned char* code and/or the requested data.

**inmemory_database**
: Class which inherits from *database.* Stores the data in memory only using standard library data types.

**disk_database**
: Class which inherits from *database.* Represents the newsgroups with folders and articles as text files, with id, title and author stored in the file name. The article text is the content of the text file.

**MessageHandler**
: Class which wraps a *Connection.* Contains functions for reading/writing single byte codes, *int* and *string.* Does not throw any errors but propagates the ones thrown by *Connection*

**Protocol**

Struct containing single byte communication codes.

**ConnectionClosedException**

Error flag struct.

**Connection**

Class containing read/write functions for single bytes, both of which throw ConnectionClosedException:s.

**Server**

Class capable of managing several connections of type *Connection*.

## Layout

The whole set up revolves around server_main and client_main. These two communicate via the *Server* and *Connection* classes. Both contain a set of functions which send and receive data, via a MessageHandler, in the way prescribed by the project description pdf, using the protocols contained in *protocol.h*.

### server_main

The server contains one database object. Whether it is a memory based or a disk based one is down to the one line of code containing the type declaration. It also contains one instance of the handout class *Server*, which for all intents and purposes simply manages a set of connections of type *Connection* and passes them along. If a connection is received, a single byte is read and passed to a switch checking for one of each of the different communication protocol codes. Each code triggers a specific function. The connection, wrapped in a *MessageHandler* is sent along to the function, which speaks to the database and sends the relevant protocol-compliant data back through the connection via the messagehandler. The functions correspond to the database functions, that is, they cover listing, creating and deleting newsgroups and articles, as well as reading specific articles. There are no database functions for checking if a newsgroup or article exists or not. Instead the database returns a message of type *unsigned char* in all cases but the "list newsgroups" function[2]. For convenience the same message protocol as for the network communication was used. It could however easily be exchanged for a separate one if a more flexible implementation was needed.

## client_main

The client is simple in its design. It contains a *Connection*, wrapped in a *MessageHandler*. It also contains a simple console consisting of a while loop over a getline function, one struct *state* for storing the minimum of information needed to function - the currently viewed newsgroup as well as a name-index maps each for the newsgroups and articles in the currently viewed newsgroup. This is in essence a state variable of the console. The string returned by the getline is parsed so that the first word[3] is stored in one string, and the rest of the original string is stored in another. This is enough to have a console where the input is on the form: ">> [command] [input]".

In each iteration of the while-loop the command string, the input string as well as the console state variable and a messagehandler are sent to a switch function. As mentioned before, there are 7 different database operations possible. The client contains two sets of these functions, two layers: The first layer of functions are triggered by switch function, which one depending on what the command and input strings contain. Some of these start off by checking whether the requested newsgroup or article is specified by name or number. They also leave room for some error handling[4]. The first layer functions trigger the second layer versions, which speak with the server. Depending on what the second layer reports back, the first layer prints either say, the requested data, or if the server says no: an appropriate message. The first and second layer of the client communicate the same way the server and database do, via a combination of data and messages. The messages used are from the messaging index contained in "protocol.h", but could easily be switched to a different index if needed.

[2]If there are none, none are sent.
[3]Separated by a space.
[4]None implemented at present.

## Example interaction

We assume that a disk_server and a client have been started and that they are speaking. The server contains a newsgroup 'ng1' with an article 'art11'.

### A. Client side

**A1. client input loop**

In the main method of the client, a *Connection* is created and sent to a *MessageHandler.* A state variable is also created. In the input loop, the line "delete newsgroup ". This string is sent to the function parse_cmd() which returns two strings: command = "delete"; input = "newsgroup".

**A2. command-input switch**

MessageHandler, command, input and state variable (references always) are sent to the function console_switch(), which looks first at the command. Having determined that the command is "delete", it looks at the input and determines that this is "newsgroup".

**A3. console input function**

MessageHandler, input and state variable are now sent to console_list_articles(). The user is now prompted to specify which newsgroup to delete (name/id). If a name is specify, the state variable will contain a name-to-index map which is used for the conversion.

**A4. server communication**

MessageHandler and newsgroup id are sent to the server communication function deleteNewsgroup(). Here the standard protocols are used to send the information according to the specification. If an error code is recieved, an error code is returned etc.

### B. Server side

**B1. disk_server connection loop**

The disk_server main method starts the disk server and an instance of the handout *Connection* handling Server class. A loop is started which waits for a *Connection* to be returned. If so, it is wrapped in a *MessageHandler* which reads the first byte using MessageHandler's read_ code().

**B2. server switch**

In our case the first byte will be the code "delete article". This triggers the corresponding database communication function.

**B3. server to database**

The *MessageHandler* as well as the database is sent to the server function delete_newsgroup(), which reads the rest of the message, receives the newsgroup index for "ng1" and speaks to the database via the corresponding database function. The database will return a code corresponding to success. The protocol compliant message for success is now sent via the *MessageHandler* to the *Connection* and back to the client.

## Guide

Usage is simple. Assuming the program has been compiled, run ./disk_server_main <port>; ./inmemory_server_main <port>; ./client_main <address> <port>. In the client type *manual* to see the available commands, or *exit* to exit.

## Conclusions

The fact that messages from the protocol.h file were used to communicate between layers on both server and client side might not be optimal. On the other hand, the only ones used for this are ack, nak, and the three error types. In both cases it would be very simple to switch to a different set of messages.

All in all, the project has been informative but time consuming. One thing that might be improved is to give a specification of the console design. This would save the student the time required to figure out a design[5]. It would also save the teacher the time required to learn all the different possible permutations[6].

---

[5]Even if this was an interesting problem to digest.

[6]Even if these may be interesting to see.