LDAP Server Documentation

Assignment for ISA

BRNO UNIVERSITY OF TECHNOLOGY FACULTY OF INFORMATION TECHNOLOGY

2017.11.11

Levente Berky

Lightweight Direct Access Protocol (LDAP) **2**
What is LDAP? 2
LDAP Message 2
Filter description 2
AND/OR filters 2
NOT filter 2
Exact Match filter 2
Substrings filter 2
Implementation **3**
Codes - header file 3
Connection 3
Database 3
Decode, Encode 3
Error 3
Logger 4
ResponseBuilder 4
Main 4
Summary 4
Testing **5**
How and where was the program tested? 5
What to test? 5
The test and the expected results 6
Summary 6
Bibliography **7**

Lightweight Direct Access Protocol (LDAP)

What is LDAP?

The Lightweight Direct Access Protocol is used by other programs to look up contacts and information about them on the server. These programs can search a certain person or group of people with variety of filters. Each entry has its attributes. These attributes can vary from server to server depending which informations do we want to store.

LDAP Message

The protocol is described in RFC documents but it is more easily understandable by UML diagrams. It uses hexadecimal encoded bytes to transfer data. The data is encoded in BER format. Usually clients can authenticate anonymously (no authentication), simple (username and password) and by SASL. After binding and authenticating, the client should send the search request which contains all the required information about the query like base, scope, time and size limit and the filter itself. The filter can contain sets of filter for logical operations AND, OR and NOT. Allows to search with exact matching string or substrings. Other filter types are not supported by this server, therefore no need to explain them. The server responses with search entries and after returned all entries or hit a limit returns a search result done message with success or the appropriate error code. Client ends the communication with an unbind request.

Filter description

AND/OR filters

The AND and OR filters contain set of filters. Each of these filters is connected with the same logical operation and that is AND or OR depending which type we got. Every of filters in this subset can contain another AND or OR filters allowing us to create any combination possible if using the NOT filter too.

NOT filter

The NOT field in the filter contains a single filter. This filter’s entries are not displayed.

Exact Match filter

The exact match field in the filter contains two fields. One is the name of the attribute to search in, the second one is the searched expression. This expression should be matched one to one.

Substrings filter

The substring filter can contain multiple substrings. Up to one times initial substring, infinite times any substring and at the end up to one times an ending substring. The order of any type substring has to be kept. The substring have to contain at least one of the named types.

Implementation

The LDAP Server is implemented in C++ and heavily relies on vectors and strings. The program contains the nex classes: Codes, Connection, Database, Decode, Encode, Error, ResponseBuilder and an helper class: Logger. In the next sections these classes will be explained in detail. However if you seek further knowledge you should look at the header files which are moderately well commented.

Codes - header file

This class contains only a header file. There you can find the request and the response structures, certain hexadecimal codes for Decoder and Encoder. The LDAP message is built from structures and enumerate types where the protocol defines it.

Connection

Contains the socket operations to set up an server. When a client connects forks a process and starts the given function. When receiving, the data is saved as string.

Database

The database contains a vector filled with custom database entry structures. The database can be filled from a file or can be returned from itself. All operation on the database returns a new database and the original is left unchanged (except toString). This behavior allows us to chain the operations but consumes more resources. AND, OR, NOT operations are implemented with the corresponding vector operations. In the database there is no size limit implemented.

Decode, Encode

These two classes uses the structures and the hexadecimal codes from the class Codes. Both classes work similar. The Decode creates structure from hexadecimal string, and the Encode creates hexadecimal string from a structure. The structures are of course LDAP messages. They have lots of functions, nearly for each data type like integer, enumeration, string and custom structures. In Decode there is an Error attribute for reporting any errors encountered when decoding the message. The message ID is checked here too. The errors are set just before an exception in thrown. Using runtime exception types we bubble the error up to the top where we catch it and we can send a response to the client.

Because the structure is not ready when the error gets to the top, there is a help function which allow us to detect the last started message type. From that information we know what type of response we should send. The error getter function allows to detect the error code and message.

Error

This class implements a simple interface for reporting any errors. Error codes are from Codes class and there is room for a custom message.

Logger
This is a static helper class and as default is not compiled. Used to test the incoming structures correctness. Only converts a structure to a printable string.

ResponseBuilder
This class makes queries to the database and builds up all LDAP message structures. From main just the received message is passed with or without the error code and the response is generated here. It can generate a set of messages if the request was an filter. The size limit is implemented here as in the Database was no room for it. When all messages are generated for the filter, returns it as a vector.

Main
In the main only the arguments are checked, default values are set and then all passed to other classes. It has one global variable, the Database. As all subprocesses will use it without changing it, it is resource friendlier to read in only once.

Summary
The program is divided into small functions and classes and everything is set to implement another filters or functionalities of LDAP server. On the other side some queries could take up lot of time and memory. The size limit is not implemented effectively. The lack of full understanding at the start caused the non perfect structure. A finite automaton would be a better solution for decoding and encoding.

Testing

The software was tested manually during the development but these test were not structured. After completing the project some goals were defined which the program now passes. In this section the method of testing will be described.

How and where was the program tested?

All tests were run on servers merlin.fit.vutbr.cz and eva.fit.vutbr.cz. Merlin was the one where the server was started and from eva with the ldapsearch util queries were sent.

Test server was running on port **38900** and with **isa2017-ldap.csv** database file supplied. This command can be run with `make run`. The file is supplied in the archive.

The test are not automatized, so after every command the results should be compared to the expected results.

What to test?

The cn, uid and mail field were tested as attributes. The next list shows which options can be tested:

- Exact Match
- Substrings
    - Initial, any, final
    - Only initial
    - Only any (one, multiple)
    - Only final
    - Initial and any
    - Initial and final
    - Any and final
- And, Or, Not
- Combinations of logical operators
    - And, And
    - And, Or
    - And, Not
    - Or, Or
    - Or, Not
    - And, Or, Not - one for NOT in AND and another for NOT in OR.
- No entry returned
- Not supported filer

For testing are used names not containing non ASCII characters for
easier evaluation (since names with UTF-8 chars are displayed in base64 encoded format in ldapsearch).

The test and the expected results

search is alias for: *ldapsearch -h merlin.fit.vutbr.cz -p 38900 -s sub -b base -x*
From these test is expected that all combinations will work.

1. **Or, And + Exact Match + Substring: only final**
    a. `search '(|(uid=xberky02)(&(mail=xklem*)(cn=*Jakub)))'`
    b. expected result: xberky02, xkleme11
2. **And, Or, Not + Substring: Init, any, final / Only Init / Init, any**
    a.
    ```
    search '
    (&
          (|
                (cn=Ud*dy*M*io) (uid=xtich*)
                (mail=xtamas01*fit*)
          )
          (!(uid=xticha04))
    )'
    ```
    b. expected result: xudvar01, xtamas01
3. **And, And + Or, Or + Substring: Init, final / any**
    a.
    ```
    search '
    (&
          (&
                (uid=*ilva*)(mail=*ilva*@*)
          )
          (|
                (| (uid=xzilva*)(uid=xberky*) )
                (cn=Zilvar Mi*)
          )
    )'
    ```
    b. expected result: xzilva02
4. **No entry**
    a. `search '(uid=xNotF00)'`
    b. expected result: Only search done response
5. **No filter supported**
    a. `search`
    b. expected result: Operations Error

Summary

Since the program passed the test we can expect that to my knowledge it is working well. The testing was not as extensive as could be but lack of time and the different purpose of the project (understanding LDAP) prevented it.

Bibliography

- Used for research:
  - RFC 4511 - https://tools.ietf.org/html/rfc4511
- The finite automaton UML which helped to understand the LDAP BER:
  - LDAP ASN.1 Codec - Apache Directory Server v1.0
    https://cwiki.apache.org/confluence/display/DIRxSRVx10/Ldap+ASN.1+Codec