

# Lightweight Directory Access Protocol (LDAP)

## Refs:

- Dr. Dobbs, March, 1999
- Netscape LDAP server docs
- U. of Michigan LDAP docs
- [www.openldap.org](http://www.openldap.org) docs
- RFCs: 1777, 1773, 1823, ...

# Directory Services

- A "directory" service is a network accessible database with limited functionality:
  - Small amount of information in each request/reply.
  - Limited functionality (as compared to a complete database system)

# Directories

- Some typical examples include:
  - telephone directories
  - lists of email addresses (or other network addresses).
- Each record is referenced by a unique key:
  - given a name, look up a phone number
  - given a name, look up a email address

# Information Structure

- Typically, the information in a directory is structured hierarchically (but it doesn't have to be).
- The structure of the data (the hierarchy) is useful in finding data and provides some (minimal) relationship between records.

# Example: DNS

- The Domain Name System is an example of a directory:
  - hierarchical structure
  - for each item there is a unique key (the hostname) and a number of attributes:
    - IP address
    - Mail exchanger
    - Host information
    - etc...

# X.500

- X.500 is a Directory Service that has been used for a while:
  - most use in Europe
  - Based on O.S.I. Protocol Stack
  - *Heavyweight* service (protocol).

# LDAP

- A number of *lightweight* front-ends to X.500 have been developed - the most recent is LDAP:
  - Lightweight Directory Access Protocol
  - Based on TCP (but can be mapped to other protocols).
  - 90% of the functionality of X.500
  - 10% of the cost

# LDAP & U. of Michigan

- LDAP originated at the University of Michigan.
- LDAP can be used as a front-end to X.500 or stand-alone.
- LDAP is now available commercially from a number of sources (including Netscape)

# LDAP definition

- RFC 1777:
  - data representation scheme
  - defined operations and mapping to requests/response protocol.
- RFC 1823: Application Programming Interface (has become a standard)

# LDAP Data Representation

- Each record has a unique key called a *distinguished name* (dn for short).
- A distinguished name (RFC 1779) is meant to be used by humans (not just computers).
- Each dn is a sequence of components.
  - Each component is a string containing an attribute=value pair.

# Example DN

CN=Dave Hollinger,

OU=Grad Student,

O=Rensselaer Polytechnic ~~University~~ Institute,

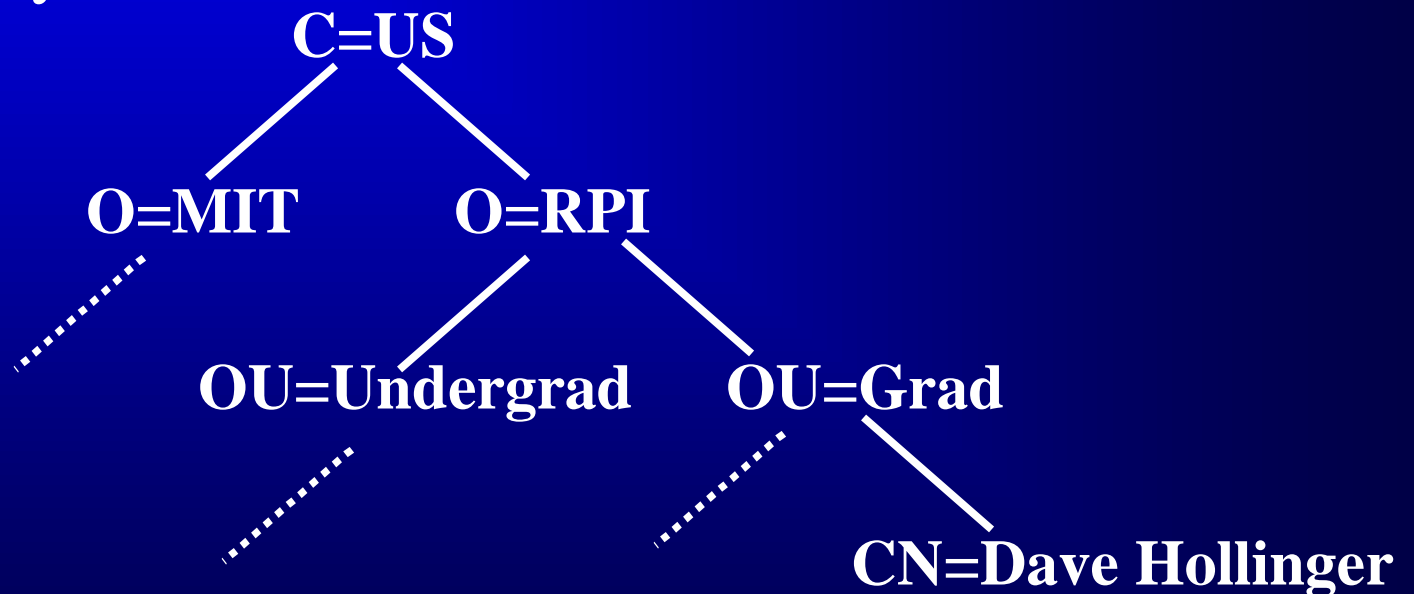
C=US

Could also be written:

CN=Dave Hollinger, OU=Grad Student, O=Rensselaer Polytechnic Institute, C=US

# Hierarchy

- Like Domain Names, the name can be interpreted as part of a hierarchy.
- The last component is at the highest level in the hierarchy:



# Component Names

- The components can be anything, but there is a standard hierarchy used (for a *global* LDAP namespace):

C	<i>country name</i>
O	<i>organization name</i>
OU	<i>organizational unit</i>
CN	<i>common name</i>
L	<i>locality name</i>
ST	<i>state or province</i>
STREET	<i>street address</i>

# Relative DNs

- Relative Distinguished Names are the individual components of a Distinguished Name (that are interpreted as relative to some position in the hierarchy).
- For example, the RDN "ou=Grad" falls in the hierarchy below "o=RPI, c=US".

# DN usage

- A Distinguished name is a key used to access a record.
- Each record can contain multiple attribute value pairs. Examples of attributes:

phone number

email address

title

home page

public key

project 3 grade

# ObjectClass

- A commonly used attribute is "objectclass".
- Each record represents an object, and the attributes associated with that object are defined according to its objectclass.
- Examples of objectclass:
  - organization (needs a name and address)
  - person (needs name, email, phone & address)
  - cookie (needs name, cost & taste index)

# Multiple Values

- Each attribute can have multiple values, for example we could have the following record:

DN: cn=Dave Hollinger, O=RPI, C=US

CN: Dave Hollinger

CN: David Hollinger

Email: hollingd@cs.rpi.edu

Email: hollid2@rpi.edu

Email: satan@hackers.org

# Project 6 Database

```
dn: course=Netprog, college=RPI
```

```
course: Netprog
```

```
objectclass: course
```

```
dn: group=Students, course=Netprog, college=RPI
```

```
course: Netprog
```

```
group: Students
```

```
objectclass: coursegroup
```

```
dn: cn=Joe Student, group=Students, course=Netprog, college=RPI
```

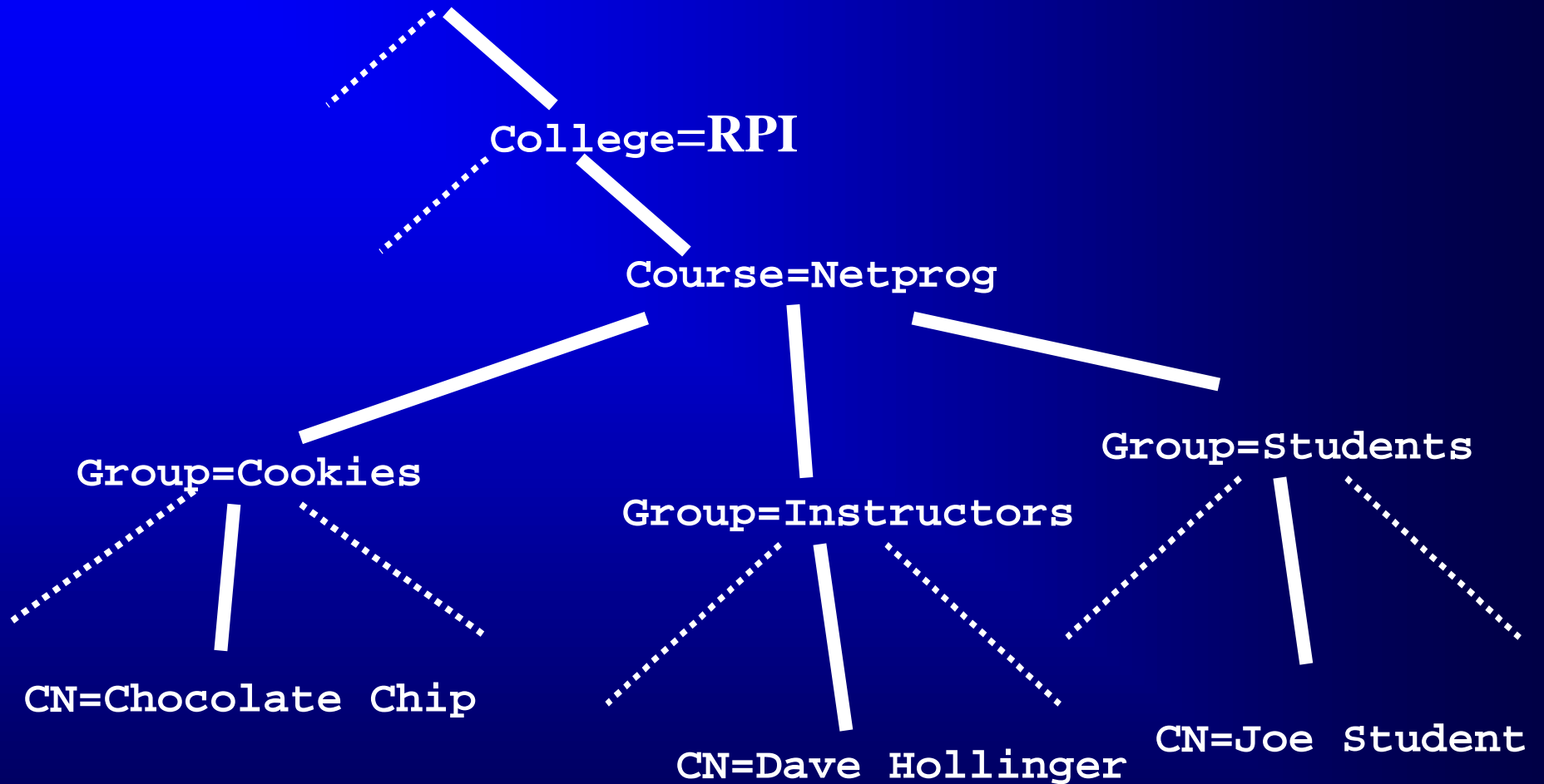
```
cn: Joe Student
```

```
email: joes@rpi.edu
```

```
objectclass: person
```

```
...
```

# Project 6 Hierarchy



# LDAP Requests

- bind/unbind (*authentication*)
- search
- modify
- add
- delete
- compare

# LDAP Protocol Definition

- The protocol is defined using in RFC 1777 using ASN.1 (abstract syntax notation) and encoding is based on BER (Basic Encoding Rules) - all very formal.
- All requests/responses are packaged in an "envelope" (headers) and include a messageID field.

# Example - LDAP bind request

- Bind request must be the first request.

BindRequest =

```
[Application 0] SEQUENCE {  
    version    INTEGER (1...127),  
    name       LDAPDN,  
    authentication CHOICE {  
        simple      [0] OCTET STRING,  
        krbv42LDAP [1] OCTET STRING,  
        krbv42DSA  [2] OCTET STRING  
    }  
}
```

# Other Requests

- Search/modify/delete/change requests can include maximum time limits (and size limits in the case of search).
- There can be multiple pending requests (each with unique messageID).
  - Asynchronous replies (each includes messageID of request).

# Search Request Parameters

**base:** DN of root of the search

**scope:** base, onelevel or subtree

**size** and **time** limits

**filter:** a search filter that defines the conditions that constitute a match.

**attributes:** a list of attributes to be returned for each match.

**attronly:** a flag that indicates whether values should be returned (or just a list of attributes)

# Search Reply

- Each search can generate a sequence of Search Response records:
  - Distinguished Name for record
  - list of attributes, possibly with list of values for each attribute.
  - Result code
- LDAP includes an extensive error/status reporting facility.

# Other Requests/Responses

- The other requests and responses are detailed in RFC1777.
- However, to write a client we don't need to know the details of the protocol, there is an API (RFC 1823) and library available!

*yippie!*

# LDAP API

- There are actually a couple of well-established APIs:
  - the original (RFC 1823) from U. of Michigan.
  - Netscape has one.
- In both cases we are spared the details of the protocol, we just call some subroutines.
- The socket stuff is handled for us.

# Writing a client

1. Open connection with a server
2. Authenticate (or authenticate if you must).
3. Do some searches/modification/deletions.
4. Close the connection

# Opening a connection

```
int ldap_bind(  
    LDAP *ld,           connection handle  
    char *dn,          who you are (your dis. name)  
    char *cred,        your credentials  
    int method)       which kind of authenticaton
```

return value is **LDAP\_SUCCESS** on success or else  
**ldap\_errno** is set to indicate the error.

# Simple bind

There are actually a bunch of `ldap_bind` functions, we can use the simplest (for P6):

```
int ldap_simple_bind(  
    LDAP *ld,           connection handle  
    char *dn,          who you are (your dis. name)  
    char *passwd)     your ldap password
```

The sample LDAP server you can use is set up so that you don't need a password (or dn) to do anything. :

```
ldap_simple_bind(l, NULL, NULL);
```

# Synchronous vs. Asynchronous

- Synchronous calls all end in "\_s"

```
ldap_simple_bind(1, NULL, NULL);
```

- Easier to use (return the result right away).

# Simple Search Query

```
int ldap_search_s(  
    LDAP *ld,           connection handle  
    char *base,         dn of the base of the search  
    int scope,          scope of the search  
    char *filter,       search filter  
    char *attrs[],     list of attributes to return  
    int attrsonly,      flag - return no values?  
    LDAPMessage **res) result of query
```

# Search Scope

- **LDAP\_SCOPE\_BASE**: search only the base for a match.
- **LDAP\_SCOPE\_ONELEVEL**: search only one level below the base.
- **LDAP\_SCOPE\_SUBTREE**: search the entire subtree below the base.

# Search Filters

- LDAP search filters are described in RFC 1960.
  - attribute=value pairs with support for boolean connectives and relational operators

- Examples:

```
"(objectclass=*)"
```

```
"(&(objectclass=Cookie)(tasteindex>=30))"
```

# Example Search

```
ldap_search_s(1,  
    "course=Netprog, college=RPI",  
    LDAP_SCOPE_SUBTREE,  
    "(cn=Joe Student)", NULL, 0, &mesg);
```

On success, mesg is a pointer to the result. To access the records in the result you have to use more of the LDAP library.

# Search Results

- The result is a list of records - you do something like this to scan the list:

```
LDAPMessage *p; char *dn;
for (p=ldap_first_entry(l,msg);
     p != NULL;
     p=ldap_next_entry(l,p)) {
    dn = ldap_get_dn(l,p);
    printf("dn: %d\n",dn);
}
```

# Attributes of each entry

- Extracting the attributes (and values) from each entry is similar - step through a list of attributes using:

```
ldap_first_attribute()
```

```
ldap_next_attribute()
```

- Example code in RFC 1823!!!

# Project 6

- Given a server and a database to play with.
- There are a number of clients that come with the OpenLDAP distribution (to play with).
- Write a client that can lookup a student's email address.
- Write a client that can change a student's email address.

# Writeup Only option

- If you don't want to write a client (15 pts), your writeup (5 pts) should contain a description of your experiences playing with the OpenLDAP server and clients.
- You must do (at least) a writeup!  
(or you lose 5 points from your homework grade)