Network Directories and their Structure . . . slide #2
Account Information . . . slide #3
Network Accounts . . . slide #4
Network Accounts — 2 . . . slide #5
Methods of achieving this . . . slide #6
Directory systems for authentication . . . slide #7
Proprietary application directories . . . slide #8
Problem with proprietary directories . . . slide #9
Why not buy Microsoft AD? . . . slide #10
LDAP— Why? . . . slide #11
LDAP Terminology . . . slide #12
What is LDAP? . . . slide #13
The LDAP Protocol . . . slide #14
Simple Search Examples . . . slide #15
Multiple Simultaneous Requests . . . slide #16
LDAP Protocol Operations . . . slide #17
Typical LDAP Exchange . . . slide #18
LDAP Encoding: BER . . . slide #19
LDAP Search Operation . . . slide #20
Search Scope . . . slide #21
The Compare Operation . . . slide #22
Add Operation . . . slide #23
Delete Operation . . . slide #24
Modify DN (Rename) Operation . . . slide #25
Modify Operation . . . slide #26
Bind Operation . . . slide #27
Command Line Utilities . . . slide #28
Common Parameters . . . slide #29
`ldapsearch` . . . slide #30
LDAP Data Interchange Format LDIF . . . slide #31
Example LDIF . . . slide #32
Update Operation in LDIF . . . slide #33
LDAP Schemas . . . slide #34
Attributes — Defined in Schema . . . slide #35
LDAP objectClass — 1 . . . slide #36
Object Class and Attributes . . . slide #37
LDAP Object Class Inheritance . . . slide #38
LDAP Object Class Type . . . slide #39
Structural Classes . . . slide #40
Entries: Selecting Object Class Types . . . slide #41
Rules for LDAP Entries . . . slide #42
Namespace of attributes . . . slide #43
Example objectTypes . . . slide #44
Want to support network login . . . slide #45
Supporting network login . . . slide #46
Authorisation as well as authentication . . . slide #47
LDAP filters . . . slide #48
RFC 2254 — 1 . . . slide #49
RFC 2254 — 2 . . . slide #50
Examples of Filters from RFC 2254 . . . slide #51
More Filter Examples . . . slide #52
Escaping Characters in a Filter . . . slide #53
Using the command line tool ldapsearch . . . slide #54
Output of this `ldapseach` without staff . . . slide #55
Get All the Results . . . slide #56
`ldapsearch` . . . slide #57
LDAP URLs: RFC 2255 . . . slide #58
`mod_auth_ldap` with Apache . . . slide #59
Authorisation of Students and Staff . . . slide #60
Other objectTypes for IVE . . . slide #61
The whole schema for IVE . . . slide #62
Case Study: ICT laboratories . . . slide #63
ICT case study . . . slide #64
ICT case study — 2 . . . slide #65
Directory Structure — 1 . . . slide #66
Directory Structure — 2 . . . slide #67
Hierarchical Directory Structure . . . slide #68
New VTC LDAP Namespace . . . slide #69
Hierarchical Directory Structure . . . slide #70
Directory Design Guidelines . . . slide #71
Designing a Schema . . . slide #72
Designing a Schema: Example . . . slide #73
References . . . slide #74

# Network Directories and their Structure

Lightweight Directory Access Protocol (LDAP)

Nick Urbanik <nicku@nicku.org>

Copyright Conditions: Open Publication License (see http://www.opencontent.org/openpub/)

A computing department

## Network Accounts

- ```
  $ ls -ln file
  -rw-rw----   1 500      500          2057 Nov  1  2000 file
  ```
- Now nicku with user ID number 500 and group ID 500 can read and write this file
- . . . But nicku with user ID number 2270 and group ID number 2270 cannot access the file at all:

  ```
  $ id
  uid=2270(nicku) gid=2270(nicku) groups=2270(nicku),14171(staff)
  ```



## Network Accounts — 2

- The user ID numbers and group ID numbers on files on a network drive are fixed
- The user ID numbers should remain unchanged for all users who read/write the network drive.



## Network Directories and their Structure

Lightweight Directory Access Protocol (LDAP)

Organising accounts in a large network

Reference book: *Understanding and Deploying LDAP Directory Services*, Second Edition, Timothy Howes, Mark Smith and Gordon Good, Macmillan, 2003.

Our library: TK 5105.595 .H69 2003

(also see references at the end of slides)



## Account Information

- The computer uses numbers to refer to users and groups
- Humans prefer to use names (like nicku)
- When you create files in your shared network drive, the client must access them using the same numbers
- The user ID numbers and group ID numbers must be the same on all computers
- Otherwise won't be able to read own files!



## Proprietary application directories

- Application-specific directories:
  - Lotus Notes
  - cc:Mail
  - Microsoft Exchange
  - Novell GroupWise
- These directories come bundled with, or, embedded into an application such as email.
- If add another such application, must manage one more directory ("$N + 1$ directory problem")
- If add another user, must add to all the directories.



## Problem with proprietary directories

- Need put the same user into many different directories
- Need maintain $N$ times the number of user accounts, where $N$ is the number of directories.
- This is just too much work.
- The accounts get out of sync.



## Methods of achieving this

- Have a *directory server* of some kind
- The directory server associates a fixed user ID number with each login ID
- . . . and a fixed group ID number for each group ID
- On NT, these are called SIDs (security IDs)



## Directory systems for authentication

- Proprietary:
  - Novell Directory Services (NDS)
  - Microsoft Active Directory (M? AD)
  - NT 4 domain
  - NIS + (Network Information System plus)
  - NIS
- Open protocols:
  - LDAP
  - Hessiod
  ```

## Why not buy Microsoft AD?

- Microsoft leverage their monopoly on the desktop to "embrace and extend" free software written by others
- Example:
  - Kerberos is a "Network Authentication Service", an IETF standard (see RFC 1510)
  - Kerberos is written by cooperating programmers round the world
  - Microsoft took Kerberos, and modified the protocol very slightly (they classified this change as a "trade secret")
  - So that MS destops can use MS Kerberos servers, but not non-MS Kerberos servers.
- Although MS claims to support standards, MS solutions are highly proprietary
- Designed to lock the user into an all-MS solution.
- Could be an expensive and insecure mistake.



## ldap— Why?

- Non-proprietary, IETF standard
  - No vendor lock-in
  - Use standard software components
- Supports authorisation as well as authentication
  - E.g., access if "staff, or year 3, group W, CSA student"
- Very general purpose: use for email, system authentication, application authentication, . . .
- Reasonably secure
- Robust
- Extensible
- Good open source implementation available at http://www.OpenLDAP.org/



## ldap Terminology

- LDAP model is *hierarchical*, i.e., tree-structured
- Each object in a directory is an *entry*
- Each individual item in an entry is an *attribute*
- Each entry has a unique full name called its *distinguished name* or *dn*
- Each entry has a short name that is unique under its parent, called its *relative distinguished name*, or *rdn*.
- The organisation of names in the directory is called the *namespace*
- An important initial task is *namespace design*



## What is ldap?

- The LDAP *protocol*, a standard Internet protocol
- Four *models*:
  - *information model*—what you can put in directory
  - *naming model*—how name directory data
  - *functional model*—what you can do with data
  - *security model*—no unauthorised access
- LDAP Data Interchange Format (LDIF), a standard text format for representing directory data
- LDAP *server software*
- *command line utilities* (`ldapsearch`, `ldapmodify`, . . . )
- LDAP API

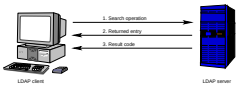


## The ldap Protocol

- LDAP is a *message-based* protocol
  - client sends one or more requests to server, one message per request
    - Each message has its own *message ID*
  - server replies with one or more replies. Each reply has message ID matching that of request.
  - Can send several messages at once; results can be out of order, no problem



## Simple Search Examples

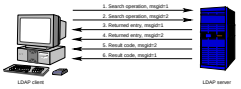



- Here a client gets one single entry from the directory

- A client gets multiple responses from the directory



## Multiple Simultaneous Requests




- A client sends multiple requests to the directory
- Note that each request has its own `msgid`
- Responses may come out of order (see last two result codes); that's okay.
  - These details are hidden from programmer by the SDK (software development kit)



## ldap Protocol Operations

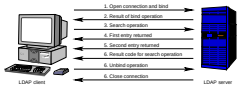
- Interrogation operations: `search`, `compare`
- Update operations: `add`, `delete`, `modify`, `modify DN` (rename)
- Authentication and control operations: `bind`, `unbind`, `abandon`

`bind` operation allows a client to identify itself sending identity and authentication credentials

`unbind` operation allows client to terminate session

`abandon` operation allows a client to tell the server it does not need the results of an operation it had requested earlier

## Typical ldap Exchange




- The *bind operation* provides a *distinguished name* (DN) and other credentials to authenticate against the directory
- The *unbind operation* is a request to disconnect



## ldap Encoding: ber

- The LDAP protocol uses the *Basic Encoding Rules*, BER to encode various data types in a platform independent way
- These are the same rules as used in SNMP
- Therefore it is not a simple text-based protocol, like HTTP or SMTP.

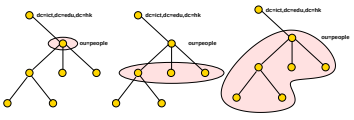


## ldap Search Operation

- Used to search for entries and retrieve them
  - This is the only way to read the directory
- Takes eight parameters, including:
  - DN of base object for search — see slide §13
  - search scope — see slide §13
  - search filter — see slide §27
  - list of attributes to return



## Search Scope




Search scope = *base* Search scope = *one* Search scope = *subtree*

- In each case, the search base is `ou=People,dc=ict,dc=edu,dc=hk`



## The Compare Operation

- Not very useful
- I use it for determining if a user belongs to a particular group
- main difference from search:
  - If compare on an attribute that does not exist in a particular entry, returns code indicating this
  - If search for an attribute that does not exist in a particular entry, then get nothing returned.



## Add Operation

- Creates a new entry, given two parameters:
  - DN of new entry
  - list of attributes and their values to put in the new entry
- Will succeed if and only if:
  - parent of new entry exists
  - no entry of same name exists
  - new entry matches requirements of schemas
  - access control allows operation



## Delete Operation

- Deletes an entry
- Takes DN of entry to delete
- Succeeds if:
  - entry exists
  - entry has no children
  - access control allows operation



## Modify dn (Rename) Operation

- Used to rename or move an entry from one place in tree to another
- Has four parameters:
  - Old DN
  - New DN
  - New RDN for entry
  - optional flag indicating whether to delete the old RDN attribute from the entry
- Succeeds if:
  - entry exists
  - new name not already used
  - access control allows operation

## Modify Operation

- Allows updating existing entry
- Can add, delete or replace attributes
- Can modify many attributes in one modify operation
- Succeeds if and only if:
  - entry exists
  - all attribute modifications must succeed
  - resulting entry obeys schemas
  - access control permits modification



## Bind Operation

- authenticates client to the directory
- Three bind types:
  - *simple bind*, where send DN and password in clear text to server
    - Need to use TLS to encrypt communication in this case
  - SASL bind
    - SASL = Simple Authentication and Security Layer
    - A standard protocol independent way of negotiating and performing authentication
  - *anonymous bind*, where send DN but no passwords
- Client can bind, perform operations, bind again, and perform other operations



## Command Line Utilities

- With OpenLDAP, the main utilities (in RH Linux, in the package `openldap-clients`) are:

  `ldapsearch` Query directory

  `ldapmodify` Perform the modify operation on an entry — see §19

  `ldapdelete` Delete an entry

  `ldapadd` Add an entry

  `ldapmodrdn` Rename an entry

  `ldapcompare` Compare operation

  `ldappasswd` Change LDAP password using LDAPv3 Password Modify (RFC 3062) extended operation
- Each one has a detailed `man` page



## Common Parameters

- All commands use the SASL (Simple Authentication and Security Layer) protocol by default
  - But won't work here:
  - . . . we use simple authentication here (we send plain text passwords over link encrypted with Transport Layer Security i.e., TLS or SSL)
- "`-x`" use simple authentication instead of SASL
- specify hostname of server with `-h`, e.g., `-h ldap.vtc.edu.hk`
- Specify a DN to bind with using `-D` (see §19)
- Specify a password on command line with `-w` ⟨*password*⟩ or interactively prompt using `-W`
  - See §19, §31 for examples



## `ldapsearch`

- Specify *base* of search with `-b`
  - Default can be specified as a line in `/etc/openldap/ldap.conf`, e.g.,

    ```
    BASE dc=tyict,dc=vtc,dc=edu,dc=hk
    HOST ldap.tyict.vtc.edu.hk
    ```
- Specify *scope* of search with `-s [base|one|sub]`
  - Default scope is subtree scope
- See §30 for more examples.



## ldap Data Interchange Format ldif

- A standard defined in RFC 2849
- Used to import, export directory data in a standard way
  - A bit like how all spreadsheets understand tab-delimited text files
- Can also specify update operations to directory entries.



## Example ldif

```
dn: uid=nicku,ou=People,dc=ict,dc=vtc,dc=edu,dc=hk
uid: nicku
cn: Nick Urbanik
givenName: Nick
sn: Urbanik
mail: nicku@nicku.org
objectClass: person
objectClass: organizationalPerson
objectClass: inetOrgPerson
objectClass: posixAccount
objectClass: top
loginShell: /bin/sh
uidNumber: 1000
gidNumber: 1000
homeDirectory: /opt/nicku
mail: nicku@nickpc.tyict.vtc.edu.hk
description: Interested in free software
```



## Update Operation in ldif

```
$ cat /tmp/update-nick.ldif
dn: uid=nicku,ou=People,dc=ict,dc=vtc,dc=edu,dc=hk
changetype: modify
replace: mail
mail: nicku@nicku.org
-
add: title
title: Lecturer in SNM
-
add: jpegPhoto
jpegPhoto:< file://tmp/pengsin.jpg
-
delete: description
-
$ ldapmodify -x -D 'uid=nicku,ou=People,dc=ict,dc=vtc,dc=edu,dc=hk \
-W -f /tmp/update-nick.ldif
Enter LDAP password:
modifying entry "uid=nicku,ou=People,dc=ict,dc=vtc,dc=edu,dc=hk"
```

## ldap Schemas

- The directory has a set of rules that determine the allowed objectclasses and attributes
- Called the *schemas*
- Can be defined in
  - ASN.1, or
  - University of Michigan style, or
  - LDAPv3 style
- Each object, and its syntax, are both defined using OIDs, as in SNMP.



## Attributes — Defined in Schema

- For each attribute, schema defines:
  - Name
  - Description
  - Permitted compare operations
  - Syntax (i.e., data type).
- LDAP server ensures that all added data matches the schema



## ldap objectClass — 1

- Each attribute belongs to one or more **objectClass**es
- objectClasses are defined in schemas
- Defines what attributes *must*, or *may* be present in an entry
- objectClass definition includes:
  - Name of objectClass
  - What subclass this is derived from
  - The type of objectClass: *structural*, *auxiliary* or *abstract*
  - Description
  - List of *required* attributes
  - List of *allowed* attributes



## Object Class and Attributes

- The entry can use all the attributes allowed in all the objectClasses.
  - See in slide §24 how LDAP attributes differ from attributes in, say, a Java class



## ldap Object Class Inheritance

- LDAP implements a limited form of object oriented inheritance
- One entry may contain many objectClasses
  - We say, "an entry belongs to many classes"
- Cannot override any schema rules defined in superior class
- Example: top ◁– person ◁– organizationalPerson ◁– inetOrgPerson
  - In `/etc/openldap/schema`, `core.schema` defines person, organizationalPerson; `inetorgperson.schema` defines inetOrgPerson
- A class derived from another class includes the attributes of its superior class(es)



## ldap Object Class Type

- objectClass has a type: *structural*, *auxiliary*, or *abstract*
- Default is *structural*
- **Structural** is for the fundamental, basic aspects of the object, e.g., **person**, **posixGroup**, **device**.
- **Auxiliary** classes place no restrictions on where an entry is stored, and are used to add more attributes to structural classes.
- **Abstract** classes are not usually created by users; the class **top** and **alias** are abstract.



## Structural Classes

- Rule of LDAP standards: if an entry belongs to more than one *structural* class, they must be related by inheritance
  - OpenLDAP 2.0.x does not implement this restriction, but OpenLDAP 2.1.x and later versions (including 2.2.x) do.
- To get around this, can either:
  - Implement a new objectClass that is of type auxiliary that allows the attributes you require—see `http://www.openldap.org/faq/data/cache/883.html`
  - Implement a new objectClass that inherits from both unrelated structural classes and use that—See `http://www.openldap.org/faq/data/cache/807.html` .



## Entries: Selecting Object Class Types

- Entries contain one or more *objectClass*es
- Choose the attributes you need
- Select the objectClasses that provide these attributes
- Add the objectClass to your entry.

## Rules for ldap Entries

- Each entry must be a member of the objectClass `top`
- Each entry must be a member of the objectClass that provides the attributes
- Exactly one objectClass should be structural, the rest auxiliary (or abstract)
  - An entry may belong to more than one structural class if all structural classes are related by inheritance



## Namespace of attributes

- There is only one namespace for attributes
- The definition of the attribute `cn` (common name) is the same for all objectClasses that support the `cn` attribute.



## Example objectTypes

- Here is the definition for person from core.schema:

```
objectclass ( 2.5.6.6 NAME 'person'
  SUP top STRUCTURAL
  MUST ( sn $ cn )
  MAY ( userPassword $ telephoneNumber $
        seeAlso $ description ) )
```

- This says a person entry *must* contain:
  - a surname (`sn`) and
  - common name (`cn`),
- and *may* contain a userPassword, a telephoneNumber, a description, and a reference to another LDAP entry.



## Want to support network login

- Does the objectClass person provide what is needed for network login?
- For network accounts, need replace (at minimum):
  - `/etc/passwd`
  - `/etc/group`
  - `/etc/shadow`
- So in addition to attributes of person, need:
  - User ID name (log in name)
  - User ID number
  - Primary group ID number
  - Gecos information (fifth field of `/etc/passwd`)
  - Home directory
  - Login shell
- Also the password aging information from `/etc/shadow`



## Supporting network login

- Use the existing objectClass `posixAccount`:

```
objectclass ( 1.3.6.1.1.1.2.0 NAME 'posixAccount'
  SUP top AUXILIARY
  DESC 'Abstraction of an account with POSIX
        attributes'
  MUST ( cn $ uid $ uidNumber $ gidNumber $
         homeDirectory )
  MAY ( userPassword $ loginShell $ gecos $
        description ) )
```

- Provides fields from `/etc/passwd`



## Authorisation as well as authentication

- Suppose you have an online web-based quiz, want only staff, or year 3, group W, CSA student to be allowed to log in.
- For this to work:
- Each person entry has attributes including:
  - `Course`, e.g., 41300
  - `classCode`, e.g., W
  - `Year`, e.g., 3
  - `acType`, e.g., STU or STF



## ldap filters

- LDAP provides a standard method for selecting authenticated users who match authorisation criteria
- The filter to select staff or students in year 3, CSA, group W is:

```
(|(acType=STF)(&(year=3)(course=41300)
(classcode=W)))
```

(This line is wrapped to fit on the slide, but normally given on one line)

- All filters are enclosed in parentheses
- Filters can be combined with OR '|', AND '&'



## RFC 2254 — 1

Find this in `/usr/share/doc/openldap-2.0.27/rfc/rfc2254.txt`

```
filter     = "(" filtercomp ")"
filtercomp = and / or / not / item
and        = "&" filterlist
or         = "|" filterlist
not        = "!" filter
filterlist = 1*filter
item       = simple / present / substring
simple     = attr filtertype value
filtertype = equal / approx / greater / less
equal      = "="
approx     = "~="
greater    = ">="
less       = "<="
```

## More Filter Examples

- Note that a filter such as `(age>21)` is not allowed.
- Use `(!(age<=21))` instead.
- Similarly, instead of `(age<21)`, use `(!(age>=21))`.
- search for all students in group X, year 3, CSA course, who enrolled this year:

  `(&(year=3)(course=41300)(classcode=W)(registrationDate=*-03))`

  Note that there is a substring match on `registrationDate` here. A substring match is like a wildcard in filename matching.



## Escaping Characters in a Filter

| Character | Escape Sequence |
|---|---|
| `*` (asterisk) | `\2A` |
| `(` (left parenthesis) | `\28` |
| `)` (right parenthesis) | `\29` |
| `\` (backslash) | `\5C` |
| `NUL` (the null byte) | `\00` |



## RFC 2254 — 2

```
present    = attr "=*"
substring  = attr "=" [initial] any [final]
initial    = value
any        = "*" *(value "*")
final      = value
attr       = AttributeDescription from Section 4.1.5 of [1]
value      = AttributeValue from Section 4.1.6 of [1]
[1] is RFC 2251.
Grammar is defined in RFC 822
```



## Examples of Filters from RFC 2254

Return all entries in the scope of the search with attribute `cn` having the value "Babs Jensen":

`(cn=Babs Jensen)`

Return all entries in the scope of the search which do `not` have the attribute `cn` with the value "Tim Howes":

`(!(cn=Tim Howes))`

Return all entries in the scope of the search which have the attibute

`(&(objectClass=Person)(|(sn=Jensen)(cn=Babs J*)))`

`(o=univ*of*mich*)`



## Get All the Results

```
$ ldapsearch -x -h ldap.vtc.edu.hk -b 'dc=vtc.edu.hk' \
"(&(department=ICT)(site=TY)(|(actype=STF)(&(year=3)
(classcode=W)(course=41300))))" cn \
| grep '^cn: ' | sed 's/^cn: //;s/^\(.\{15\}\).*/\1/' | sort | column
Andy LAI        C M Ho          LEE HUNG KIN    SIU CHONG PUI
CHAN CHIN PANG  Curtis H.K. Tsa LEE KOON HUNG K SIU WAI CHEUNG
CHAN Kwok Kam   Esther YUEN     LEUNG KAM SHEK  Stella Chu
CHAN KWOK KEUNG Eva Chung       LI Kim Wah      TAM CHI HO
CHAN SHIU CHUAN FONG CHI KIT    LI Siu Kai      TAM Kin Fai
CHAN TAI HING   Henry Leung     LI Yuet Cheung  TSANG KWOK TUNG
CHAN TAI MING R HO CHUN WAH     MA Hei Man      TSO Yee Yee
Charles Wu      HO KIM MAN ALBE MA SUI WAH      WONG Chi Man
CHEUK Suk Lai   Josephine Wan   MICHAEL LEUNG   WONG Hoi Shan
CHEUNG KAM HOI  Karl Leung      MO Hoi Yu       WONG Siu Fai
CHEUNG SAI MING Ken LI          MONTAGUE NIGEL  WONG WAI YIP FR
CHIK FUNG YING  Kit K. KO       NG HOI KOW      Wong Y.L. Lawre
CHIU SUET FAN J LAI HING BIU    NG SZE CHIU EDD WOO HUNG CHEUNG
Chou Siu Chuen  LAI Man Chiu    Nick Urbanik    WOO Kin Fan
CHUNG Ming Kit  LAM Lai Hang    PATRICK K.S. TO YIM KWOK HO
CHU SHING TSU J LAU KWOK ON     POON Chun Chung Y.K. Leung
Clarence Lau    LAU Siu Ying    Rick Liu
Clarence Lo     LAW Yuk Woon    SCOTT ALBERT HE
```



## `ldapsearch`

- Needs the `-x` option to work here
- Check ssl works with the `-ZZ` option
- Can "bind" as a user to get all the info you are allowed to see after binding:

  ```
  $ ldapsearch -x -W -D \
  "uid=nicku,ou=People,dc=tyict,dc=vtc,dc=edu,dc=hk" \
  '(uid=nicku)'
  ```

- Can then see own passwords



## Using the command line tool ldapsearch

- `$ ldapsearch -x -h ldap.vtc.edu.hk \`
  `-b "dc=vtc.edu.hk" \`
  `"(&(department=ICT)(site=TY)`
  `(|(acType=STF)`
  `(&(year=3)(course=41300)(classcode=W))))" cn`
- The result is a list of all the DNs that match the filter, with the students' and staff names.
- Can filter out the DNs and blank lines by piping the command though `grep '^cn:' | sort`



## Output of this `ldapseach` without staff

```
cn: CHAN Kwok Kam
cn: CHEUK Suk Lai
cn: CHUNG Ming Kit
cn: LAI Man Chiu
cn: LAM Lai Hang
cn: LAU Siu Ying
cn: LAW Yuk Woon
cn: LI Kim Wah
cn: LI Siu Kai
cn: LI Yuet Cheung
cn: MA Hei Man
cn: MO Hoi Yu
cn: POON Chun Chung
cn: TAM Kin Fai
cn: TSO Yee Yee
cn: WONG Chi Man
cn: WONG Hoi Shan
cn: WONG Siu Fai
cn: WOO Kin Fan
```

## Authorisation of Students and Staff

- We need a new schema to support the required attributes
- We create three new objectClasses and associated attributes:
- The first is common to students and staff:

```
objectclass ( 1.3.6.1.4.1.11400.2.2.1 NAME 'institute'
  SUP top AUXILIARY
  DESC 'Any person in the institute, staff or student'
  MAY ( acOwner $ acType $ answer1 $ answer2 $
        answer3 $ batchUpdateFlag $ department $
        site $ instituteEmail ) )
```



## Other objectTypes for IVE

- Then on top of this, we have attributes for students:

```
objectclass ( 1.3.6.1.4.1.11400.2.2.2 NAME 'student'
        SUP top AUXILIARY
        DESC 'A student in the institute'
  MAY ( academicYear $ award $ classCode $ course $
        courseDuration $ FinalYear $
        registrationDate $year $
        fullPartTime ) )
```

- . . . and staff:

```
objectclass ( 1.3.6.1.4.1.11400.2.2.3 NAME 'staff'
        SUP top AUXILIARY
        DESC 'A staff member of the insitute.'
  MAY ( titleDes $ employerID ) )
```



## ldap URLs: RFC 2255

- Have the form:
- `ldap://`⟨*host*⟩`:`⟨*port*⟩`/`⟨*base*⟩`?`⟨*attr*⟩`?`⟨*scope*⟩`?`⟨*filter*⟩

```
ldapurl    = ldap://" [hostport] ["/"
                   [dn ["?" [attributes] ["?" [scope]
                   ["?" [filter] ["?" extensions]]]]]]
```

- The ⟨*base*⟩ or `dn` is the distinguished name of the starting entry for your search.
- ⟨*scope*⟩ is one of `base`, `one` or `sub`
- Examples:

```
ldap://ictlab/ou=People,dc=tyict,dc=vtc,dc=edu,dc=hk?uid?one?(uid=nicku)
```



## `mod_auth_ldap` with Apache

- `mod_auth_ldap` is part of the `httpd` RPM package on Fedora Core 1.
- Here we allow staff or students from group W, year 3 CSA to access the web pages under `http://hostname/group-w/` if the user provides a correct password:

```
<Location "/group-w">
     AuthType Basic
     AuthName ''\LDAP authentication to class W only"
     AuthLDAPURL ldap://ldap.tyict.vtc.edu.hk/
ou=People,dc=tyict,dc=vtc,
dc=edu,dc=hk?uid?one?(|(acType=STF)(\&(course=41300)
(classCode=W)(year=3)))
     require valid-user
</Location>
```

- See http://httpd.apache.org/docs-2.0/mod/mod_auth_ldap.html, and also http://httpd.apache.org/docs-2.0/mod/mod_ldap.html for manual.



## The whole schema for IVE

- The whole schema can be seen here: `http://nicku.org/oids/institute.schema`



## Case Study: ICT laboratories

- Old system:
- An ancient DEC Alpha running NIS
- Hardware insufficient for demand
- *Very* expensive maintenance, stopped paying
- Technician reported a hardware failure close to first day of term
- New system:
- We were planning to introduce LDAP authentication gradually
- Failure required planning move faster
- Needed to maintain old legacy accounts, plus introduce new accounts



## ICT case study

- We chose OpenLDAP on Linux
- Running on an Acer Altos dual CPU P-III
- Migrated from the NIS using the migration scripts provided with OpenLDAP
- Migrated from the VTC LDAP accounts using a Perl program, written (quickly!) for the purpose,
- Uses the `Net::LDAP` Perl modules

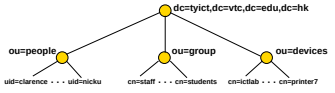


## ICT case study — 2

- After migrating the legacy accounts, and creating new accounts for staff, full and part time students, had more than 5000 accounts
- The LDAP server was using a high CPU load
- Was able to solve this using caching:
- Use `nscd` (name service caching daemon) on client
- Use memory in server to increase local cache size drastically.
- CPU load reduced to a very acceptable level.

## Directory Structure — 1

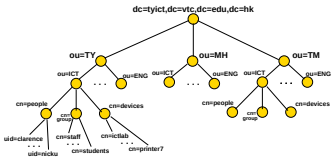
- The ICT LDAP server namespace design:






## Hierarchical Directory Structure

- This directory structure is hierarchical:





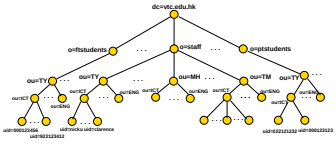

## Directory Structure — 2

- We chose a fairly flat directory structure
- Recommended by reference, pages 239, 249.
- Reason: flexibility:
- allows for change without major reorganisation of data.



## New VTC ldap Namespace

- This new VTC LDAP namespace was introduced in April 2003:






## Hierarchical Directory Structure

- This is an alternative data arrangement
- Divide into different campuses
- Advantage: can easily delegate management to local campus
- But: suppose ENG changes to EE?
- Suppose staff move from one department to another?
- Suppose equipment is transferred?
- Not only need change the attributes in the entry, but also move the entry.
- Overall, a flatter structure is easier to manage.



## Directory Design Guidelines

- Design as flat as possible given constraints:
- Replication
- Access Control
- Limitations of directory software
- Requirements of applications that use the directory



## Designing a Schema

- After selecting the schema attributes needed for your application, you may find that not all are available with the server
- Search web for more schemas
- If none provide all you need,
- Select a suitable structural base class
- Create an auxiliary class to be used with the base class
- Define the objectClass and its attributes



## Designing a Schema: Example

- For our ICT LDAP server, we use enough attributes to be able to log in
- But we also want to select users on the basis of course, year, class
- Want to add these attributes to the existing objectClasses
- Create three object classes:
- Institute
- Student
- Staff

## References


- *LDAP System Administration*, Gerald Carter, ISBN 1-565-92491-6, O'Reilly, March 2003
- *Understanding and Deploying LDAP Directory Services* (2nd Edition), Tim Howes, Timothy A. Howes, Mark C. Smith, Gordon S. Good, ISBN: 067232168, Publisher: Addison Wesley Professional, May 2, 2003
- *Understanding and Deploying LDAP Directory Services*, Timothy Howes, Mark Smith and Gordon Good, Macmillan, 1999. Our library: TK 5105.595.H69 1999
- *LDAP Programming, Management and Integration*, Clayton Donley, ISBN: 1-930110-40-5, Manning Publications, 2003
- *LDAP Directories Explained: An Introduction and Analysis*, Brian Arkills, ISBN 0-201-78792-X, Addison-Wesley, 2003.
- *Understanding LDAP Redbook* (registration required): http://www.redbooks.ibm.com/abstracts/sg244986.html
- *LDAP Implementation Cookbook Redbook* (registration required): http://www.redbooks.ibm.com/abstracts/sg245110.html
- *Implementing LDAP*, Mark Wilcox, Wrox Press, 2000