COT 4600 Operating Systems Fall 2009

Dan C. Marinescu

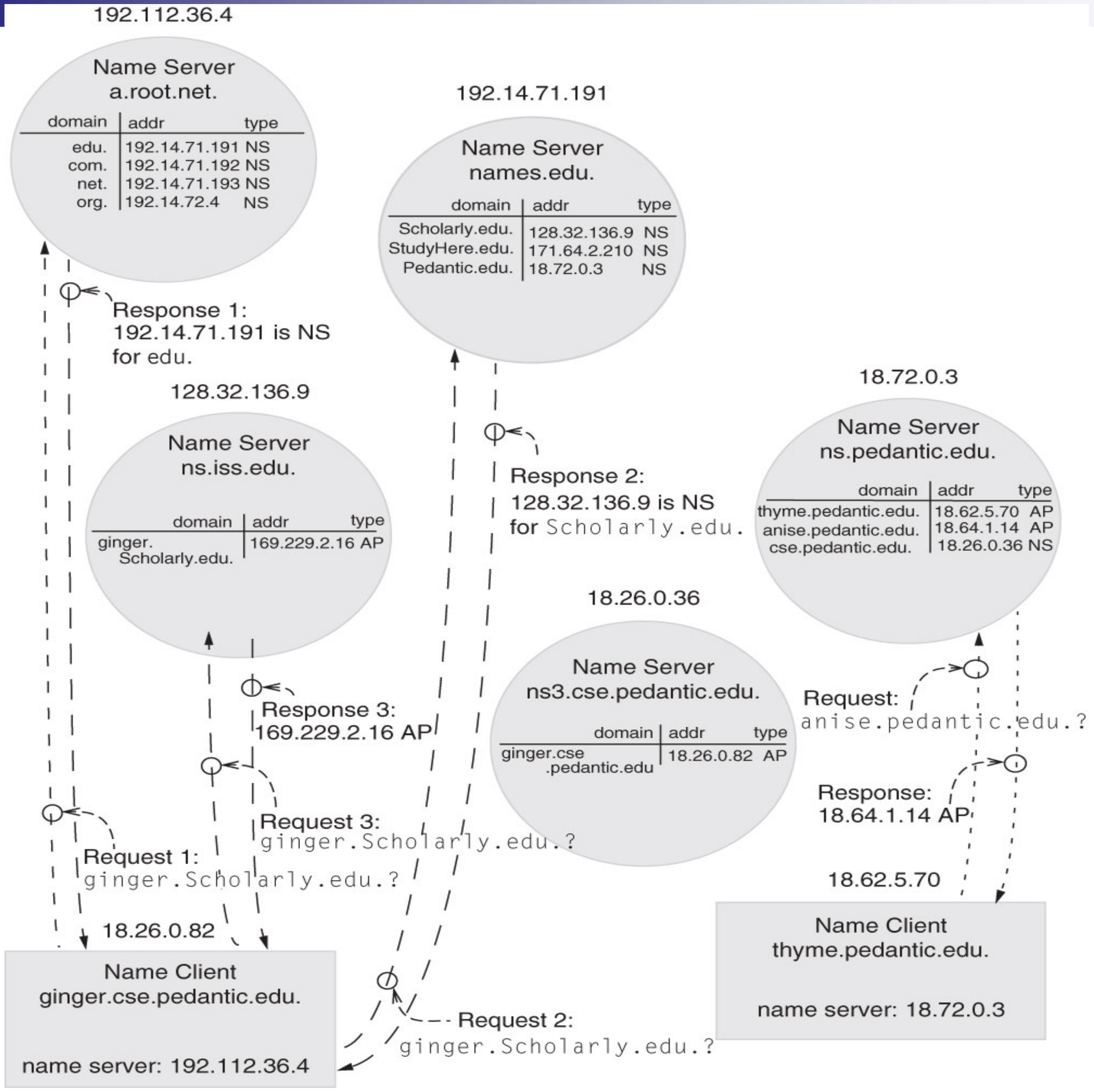
Office: HEC 439 B

Office hours: Tu-Th 3:00-4:00 PM

Lecture 15

- Last time:
  - Virtual Memory
- Today
  - Domain Name Service (DNS)
  - Network File System (NFS)
  - Question and answers about the project and the midterm
- Next Time:
  - Midterm

Important: phase 2 of the project is now due on Thursday, October, 22, together with HW4.


192.112.36.4
Name Server a.root.net.
192.14.71.191
Name Server names.edu.
Response 1: 192.14.71.191 is NS for edu.
128.32.136.9
Name Server ns.iss.edu.
Response 2: 128.32.136.9 is NS for Scholarly.edu.
18.72.0.3
Name Server ns.pedantic.edu.
18.26.0.36
Name Server ns3.cse.pedantic.edu.
Response 3: 169.229.2.16 AP
Request: anise.pedantic.edu.?
Response: 18.64.1.14 AP
Request 3: ginger.Scholarly.edu.?
Request 1: ginger.Scholarly.edu.?
18.26.0.82
18.62.5.70
Name Client ginger.cse.pedantic.edu.
name server: 192.112.36.4
Request 2: ginger.Scholarly.edu.?
Name Client thyme.pedantic.edu.
name server: 18.72.0.3

Name Server a.root.net.

domain	addr	type
edu.	192.14.71.191	NS
com.	192.14.71.192	NS
net.	192.14.71.193	NS
org.	192.14.72.4	NS

Name Server names.edu.

domain	addr	type
Scholarly.edu.	128.32.136.9	NS
StudyHere.edu.	171.64.2.210	NS
Pedantic.edu.	18.72.0.3	NS

Name Server ns.iss.edu.

domain	addr	type
ginger. Scholarly.edu.	169.229.2.16	AP

Name Server ns.pedantic.edu.

domain	addr	type
thyme.pedantic.edu.	18.62.5.70	AP
anise.pedantic.edu.	18.64.1.14	AP
cse.pedantic.edu.	18.26.0.36	NS

Name Server ns3.cse.pedantic.edu.

domain	addr	type
ginger.cse .pedantic.edu	18.26.0.82	AP

The virtues of DNS

- Distributed responsibility → any DNS name server may act as a naming authority and
  - add authoritative records (see example on the previous slide, the right diagram)
  - create lower-level naming domains; e.g., UCF can create EECS, EECS can create ComputingFrontiers, etc.
- Robustness→
  - High level of replication of the name servers
    - There are some 80 replicas of the root name server
    - Each organization with a name server has 2-4 replicas
  - Stateless name servers → does not maintain any state, its public interface is idempotent
  - A DNS server is a dedicated computer running a relatively simple code, thus less likely to fail

More virtues and some problems of DNS

- Flexibility →
  - The same name may be bound to several IP addresses. Needed to
    - ensure replication of services
    - improve performance → see for example the content delivery services provided by akamai
  - Allows synonyms
    - a computer may appear to be in two different domains
    - Indirect names
- Lack of authentication → DNS does not use protocols to authenticate the response to a DNS request. One can impersonate a DNS server and provide a fake response.
- Does not guarantee accuracy →a DNS cache may hold obsolite information

The Network File System

- Developed at Sun Microsystems in early to early 1980s.
- Application of the client-server paradigm.
- Objectives:
  - Design a shared file system to support collaborative work
  - Simplify the management of a set of workstations
    - Facilitate the backups
    - Uniform, administrative policies
- Main design goals
  1. Compatibility with existing applications → NFS should provide the same semantics as a local UNIX file system
  2. Ease of deployment → NFS implementation should be easily ported to existing systems
  3. Broad scope → NSF clients should be able to run under a variety of operating systems
  4. Efficiency → the users of the systems should not notice a substantial performance degradation when accessing a remote file system relative to access to a local file system

NFS clients and servers

- Should provide transparent access to remote file systems.
- It mounts a remote file system in the local name space → it perform a function analogous to the MOUNT UNIX call.
- The remote file system is specified as *Host/Path*
  - *Host* → the host name of the host where the remote file system is located
  - *Path* → local path name on the remote host.
- The NFS client sends to the NFS server an RPC with the file *Path* information and gets back from the server a file handle
  - A 32 bit name that uniquely identifies the remote object.
- The server encodes in the file handle:
  - A file system identifier
  - An inode number
  - A generation number

Why file handles and not path names

-------------------------------- Example 1 ---

Program 1 on client 1 | Program 2 on client 2

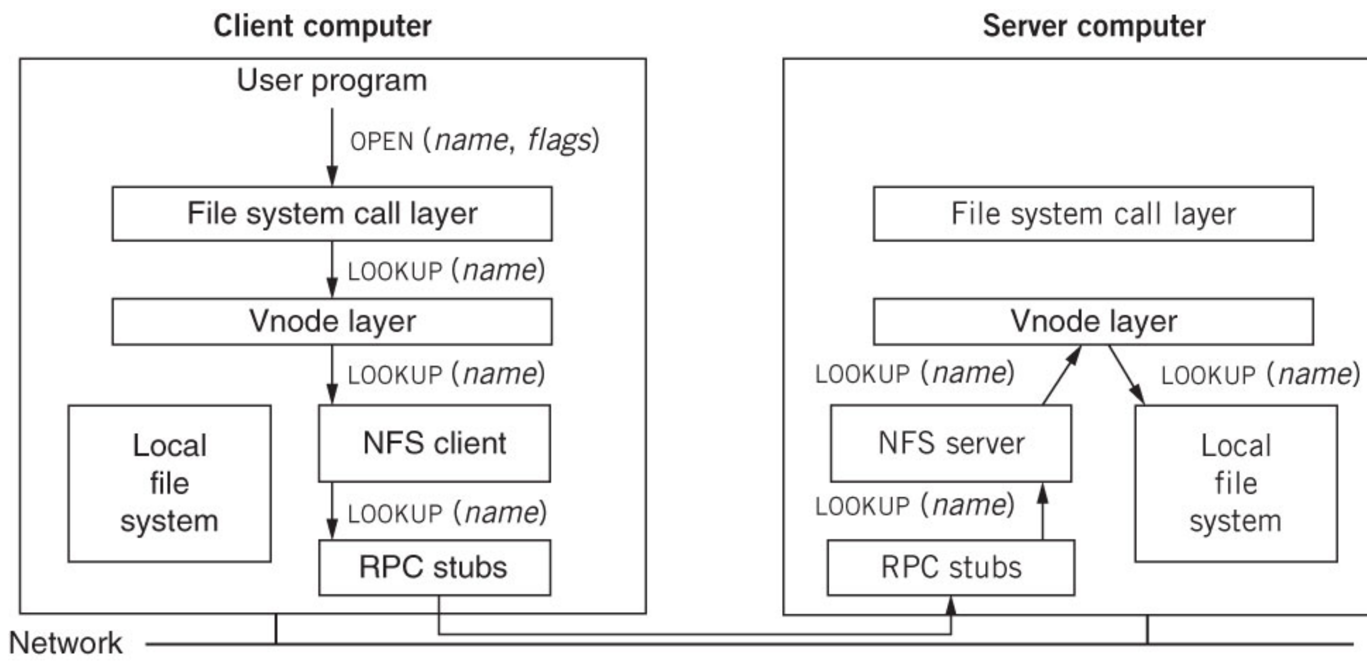
```
Program 1 on client 1                    Program 2 on client 2
  CHDIR ('dir1')
  fd ← OPEN("f", READONLY)
                                          RENAME('dir1','dir2)
                                          RENAME('dir3','dir1')
  READ(fd,buf,n)
```

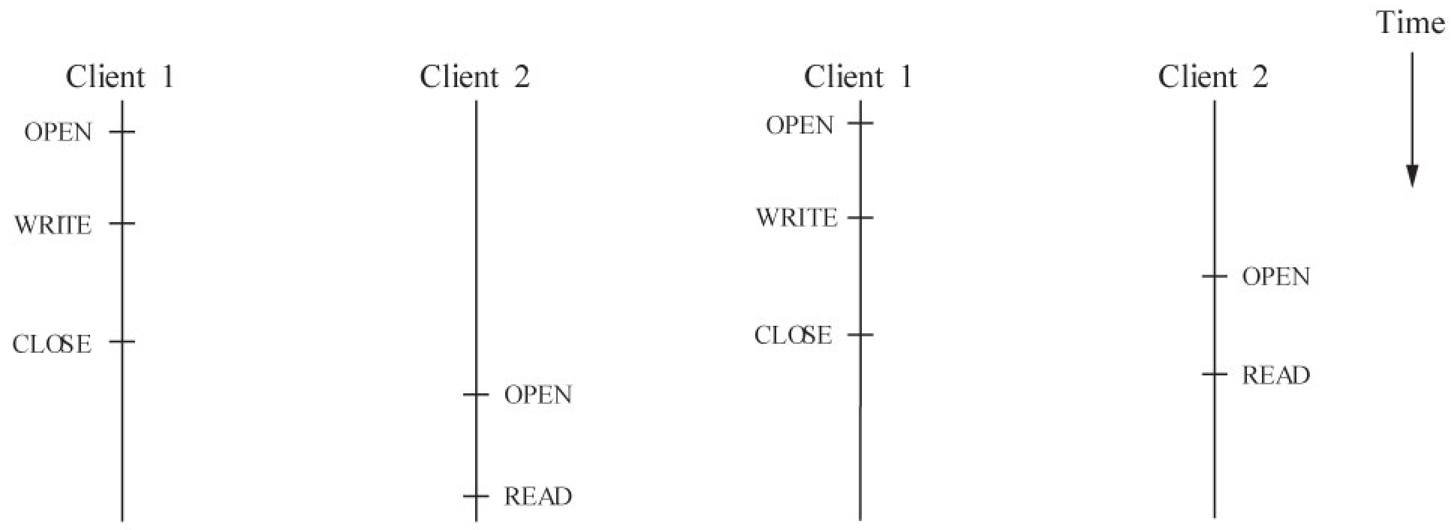
To follow the UNIX specification if both clients would be on the same system client1 would read from dir2.f. If the inode number allows the client 1 to follw the same semantics rather than read from dir1/f

---------------------------------- Example 2 --

```
                                          fd ← OPEN("file1", READONLY)
UNLINK("f")
fd ← OPEN("f",CREATE)
                                          READ(fd,buf,n)
```

If the NFS server reuses the inode of the old file then the RPC from client 2 will read from the new file created by client 1. The generation number allows the NSF server to distinguish between the old file opened by client 2 and the new one created by client 1.


Client computer
User program
OPEN (*name*, *flags*)
File system call layer
LOOKUP (*name*)
Vnode layer
LOOKUP (*name*)
Local file system
NFS client
LOOKUP (*name*)
RPC stubs
Server computer
File system call layer
Vnode layer
LOOKUP (*name*)
LOOKUP (*name*)
NFS server
Local file system
LOOKUP (*name*)
RPC stubs
Network



Case 1: READ observes last WRITE

Case 2: READ may observe last WRITE or not