

Plan 9: Not (Only) A Better UNIX

Jim Huang (黃敬群) <jserv@0xlab.org>

Developer, 0xlab – <http://0xlab.org/>

May 15, 2012 / JuluOSDev

Rights to copy

© Copyright 2012 **0xlab**

<http://0xlab.org/>

contact@0xlab.org



Attribution – ShareAlike 3.0



You are free

- ✓ to copy, distribute, display, and perform the work
- ✓ to make derivative works
- ✓ to make commercial use of the work

Corrections, suggestions, contributions and translations are welcome!

Latest update: May 15, 2012

Under the following conditions

-  **Attribution.** You must give the original author credit.
-  **Share Alike.** If you alter, transform, or build upon this work, you may distribute the resulting work only under a license identical to this one.
- For any reuse or distribution, you must make clear to others the license terms of this work.
- Any of these conditions can be waived if you get permission from the copyright holder.

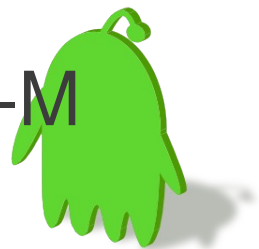
Your fair use and other rights are in no way affected by the above.

License text: <http://creativecommons.org/licenses/by-sa/3.0/legalcode>



Beyond this presentation

- <http://sync.in/16W2nkk2vZ>
- Design our own world-dominating OS.
- Discuss Plan 9, an OS that many people thought should have dominated the world.
- Meeting of Open Source Users in Tainan (5/19)
<http://mosut.org/>
- COSCUP 2012
 - OSDev Group
 - Call for Participation!
- NCKU Lecture (Sep 2012)
 - Learn system programming for ARM Cortex-M
 - RTOS, toolchain, system modeling



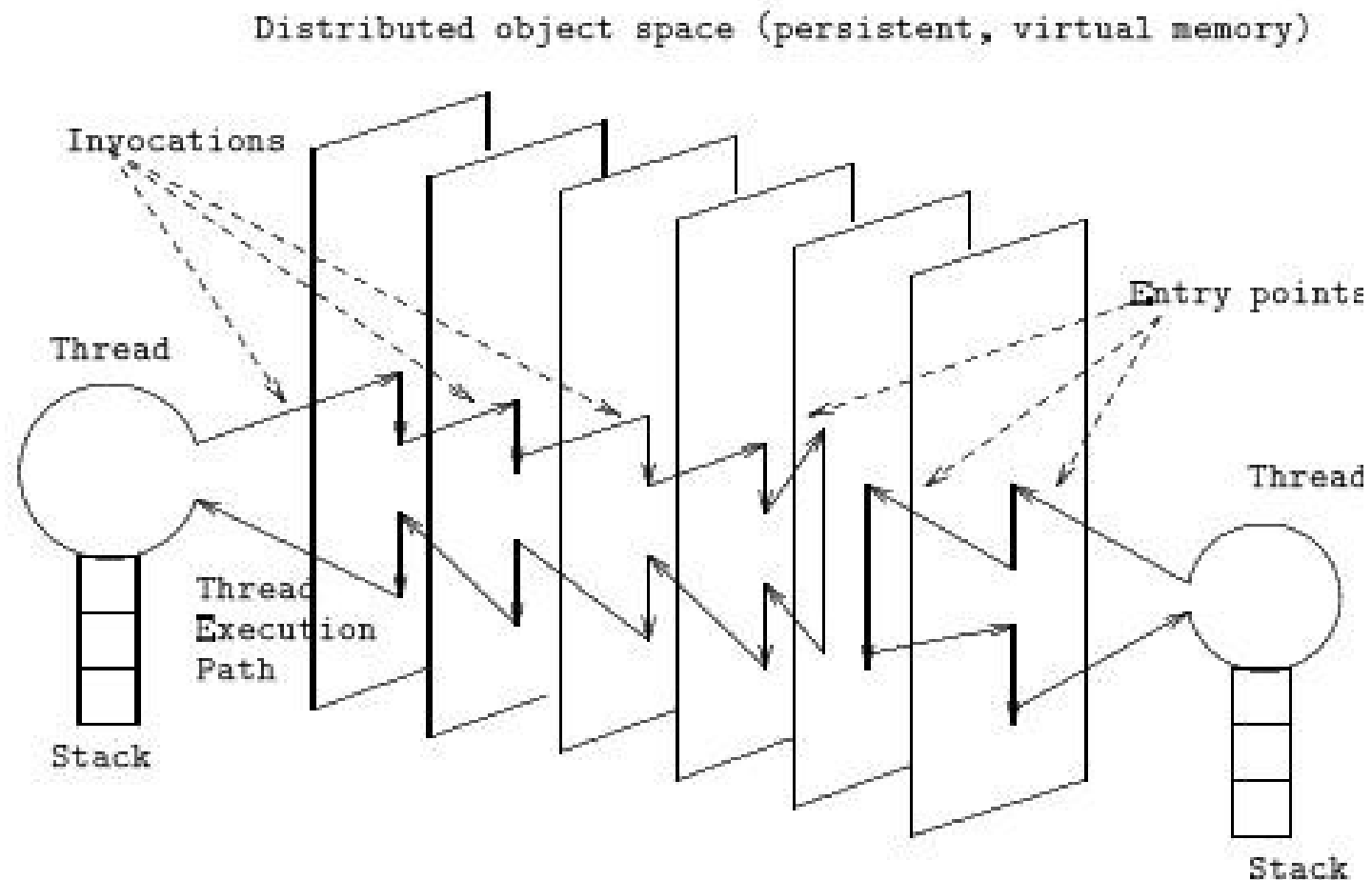
History always inspires us

- 「雲端作業系統」 is not a new word.
 - Clouds!
- Clouds is actually 2 separate operating systems
 - Clouds V1, based on VAX kernel (1986)
 - Clouds V2, based on Ra kernel (~1989)
- Purpose: Support distributed research at Georgia Institute of Technology



Clouds

- Based on the object/thread paradigm
- At OS level there is only one type of object – *clouds*
- Ra kernel implements persistent virtual memory
 - Threads travel through objects
 - Entry points



「十年前你能想像現在人們在爭論
*BSD 與 Linux 哪個使用者體驗比較
好嗎？」

from clkao



In memory of the heros...

- Dennis M. Ritchie (dmr; 1941-2011)
 - UNIX, Plan 9
 - Related talk: Plan 9, May 15, 2012
- Jochen Liedtke (1953-2001)
 - L4 microkernel
 - Related talk: L4 Microkernel, Feb 7, 2012



Some Areas for discovery

- Networking / Sockets
- Threading
- File I/O
- Security
- Scheduling
- Windowing System / User Interface
- Examples:
 - Eliminate all non-unicode char *'s.
 - Add an “e” “creat()” system call (Ken Thomson)
 - Copy / paste: an open problem (consolidate to a centralized copy/paste server).



Agenda

- (1) Design Concepts
- (2) Essential Plan 9
- (3) Renaissance



Design Concepts



Unix Origins (~-1969)

- Bell Labs has a long history in early operating systems, for example BE-SYS for IBM 709x machines; Multics
- Ken Thompson wanted to write a computer operating system by the 1960s:
 - Explore structures for building OS
 - Build something for our own group to use
- Fundamental idea: a good way to represent data (disk files)
- First steps for Unix:
 - find concrete representation for data on disk
 - define access methods to data
- Earliest Unix simply tried to build some superstructure to test Thompson's ideas



Early Ideas about Unix (1969-1972)

- Much was inherited, especially from Multics project
- Files contain just a sequence of bytes:
 - interpretation is up to applications
 - optimization of access is up to operating system
 - preference is for files with readable text, not binary
- Files are named in a hierarchical, tree-like name space, e.g.

`/usr/dmr/japan/japanslides.ppt`



File System access operations

- Basic operations are very simple:
 - `handle = open(filename, read-or-write-mode)`
 - `handle = create (filename, protection-mode)`
 - `read(handle, buffer, nbytes)`
 - `write(handle, buffer, nbytes)`
 - `close(handle)`
 - `seek(handle, place)`



Hierarchical Names

```
/
-- source
  -- shell
    sh1.c
    sh2.c
  ...
-- usr
  -- ken
  ...
  -- dmr
    -- japan
      japanslides.ppt
    ...
  ...
-- bin
  Sh
  ...
-- ...
```



Names for other things

- Unix introduced a consistent abstraction: names for I/O devices as part of the same hierarchy, for example
 - `/dev/tty23` (*name of terminal 23*)
 - `/dev/disk/disk03` (*name of a whole disk*)
 - `/dev/mem` (*name for main memory*)
- These really exist in the file system and the same protection and ownership properties are applied as with ordinary files
- To the extent possible, they look like the same byte streams as plain disk files
- Some (like terminals and networks) have some special operations that apply: “I/O controls”



Remote file system

- By 1980s, Sun (with NFS), AT&T Computer Systems (RFS), and Bell Labs Research (NetA) were building “remote file systems”
 - others pioneered also, like Xerox
 - Using RPC (remote procedure call) mechanisms, attach file system hierarchies on other machines to a local machine
- Important generalization: approach to a distributed system transparent to applications
- Some problems:
 - How usable are remote I/O devices?
 - *Some problems with protocols and blocking devices*
 - How usable are remote resources?
 - *Interworking across machine architectures, e.g. Differences in byte ordering*





Next step: Plan 9 (1990-2003)

- Designed in mid 80's by many of the creators of UNIX (Rob Pike, Ken Thompson, Brian Kernighan, Dennis Ritchie)
- Paradigm shift in mid 1980's
 - From: Large centralized timeshared computers
 - To: Networks of smaller personal machines (UNIX 'workstations')
- Original idea: define a remote protocol (called 9P) to talk to the file system
- Unix began with a structural idea:
 - How to represent a local file system?
- Plan 9 began with this structural idea:
 - How to talk to resources (local or remote)?



“Plan 9 from Bell Labs” based on “Plan 9 from Outer Space.”



Worst movie ever made
(by Ed Wood)



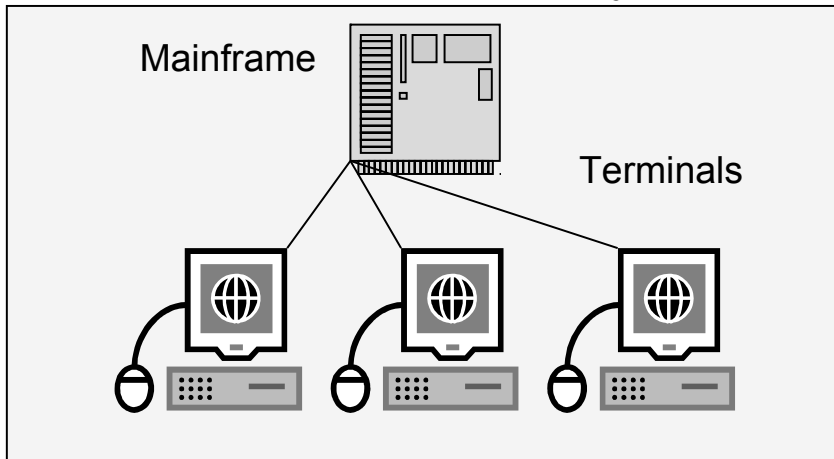
Plan 9 from Bell labs

- Designed in mid 80's by many of the creators of UNIX (Rob Pike, Ken Thompson, Brian Kernighan, Dennis Ritchie)
- Inferno is a light and embedded version of Plan 9
 - sold to Vita Nuova Holdings and has been released under GPL/MIT license.
- “Build a UNIX out of little systems”
 - not “a system out of a lot of little Unixes”
 - Take the good things
 - Tree-structured file system
 - “Everything is a file” model
 - Toss the rest (ttys, ioctl(), ***signals!***)

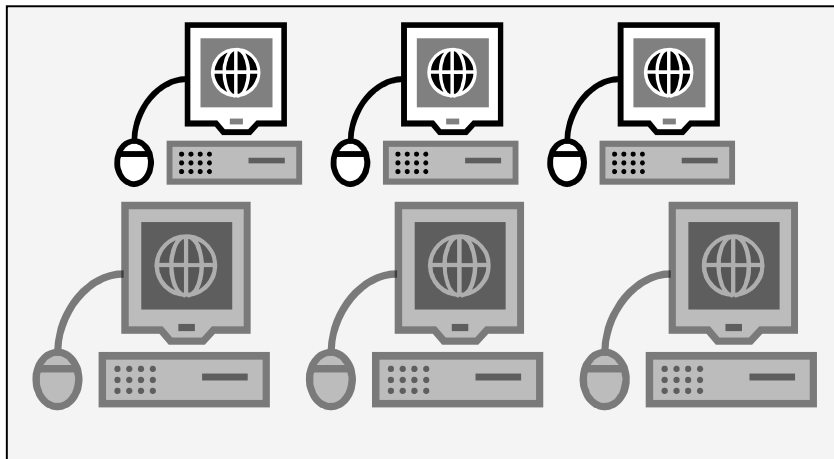


Plan 9 Goals

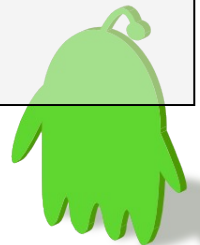
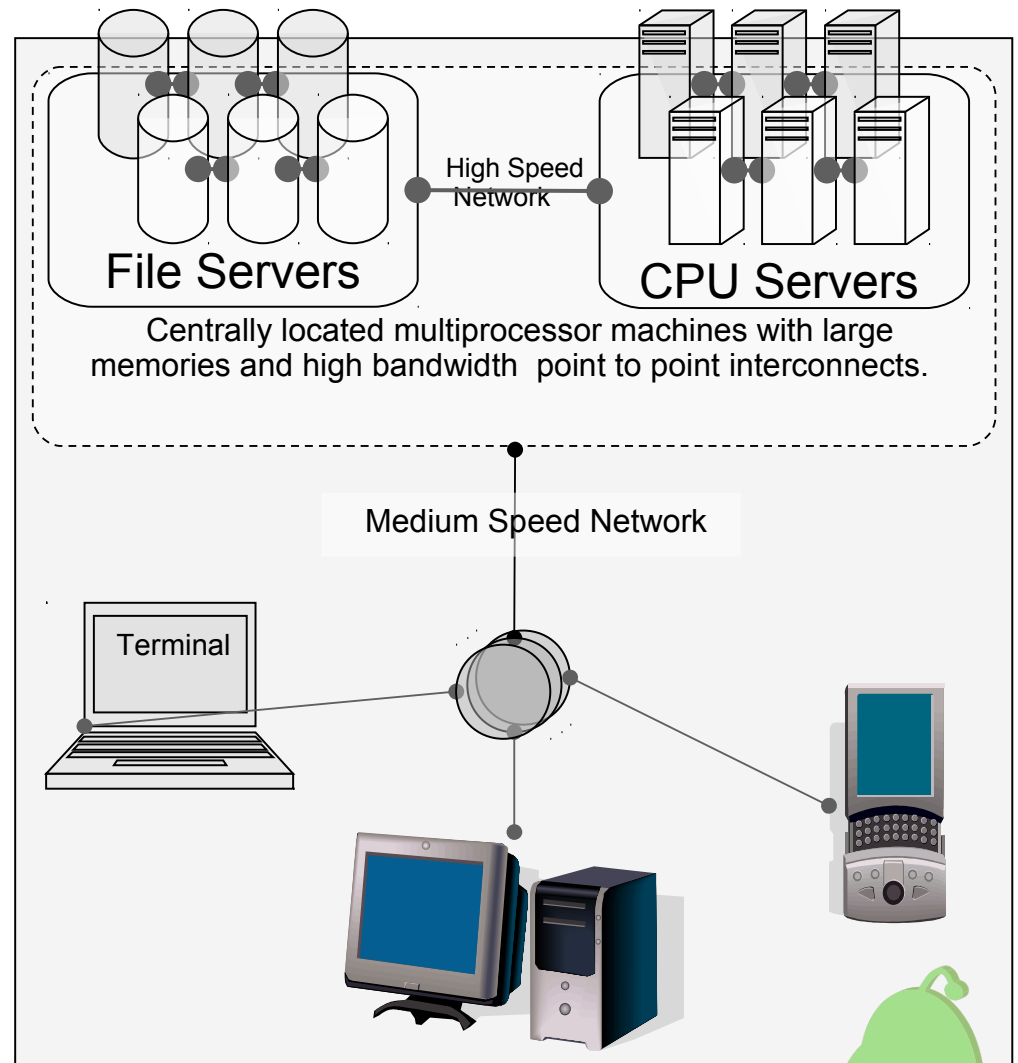
Centralized Time shared Systems



Personal Workstation Systems



Plan 9 Distributed Operating System



Plan 9 Design Principles

- Everything is a file”
 - Standard *naming system* for all resources: pathnames
- “Remote access” is the common case
 - Standard *resource access protocol*: “9P” (“9P2000”)
 - Used to access any file-like thing, remote or local
- Personal namespaces
 - Naming ***conventions*** keep it sane
- A practical issue: Open Source
 - Unix source not available at “Bell Labs”, its birthplace!



Plan 9 Assumptions

- Reliable machine-room **file servers**
- Shared-memory multiprocessor **cycle (CPU) servers**
 - Located near file servers for fast access
- Remote-access workstation **terminals**
 - Access your **view** of the environment
 - Don't **contain** your environment
 - Disk is optional
 - Typically used for faster booting, file cache
 - “Root directory” is located on your primary file server



Everything is a File really.

- In many early operating systems, different types of block devices had different API's.
- One major innovation in UNIX was the single file I/O API for multiple devices.
- Plan 9 treats virtually all devices and services as file-based services:
 - telnet
 - ftp
 - nfs



Notable architecture

- All system interfaces through file system. (pre-Linux)
- Workstation - independent working environment
 - aggregated resources, remote and local.
- /proc
 - All processes are visible as files
- /net
 - All network traffic read/written through file system
- Can import POSIX apps, emulate Berkeley socket interface through APE (ANSI/POSIX Environment)



Example: Open a socket

```
% cat /net/tcp/clone/ctl
```

```
44
```

```
% cd /net/tcp/44
```

```
% echo connect 128.2.194.80!79 > ctl
```

```
% echo ping > data
```

```
% cat data
```

```
pong
```



Namespace

- Any file resource can be “mounted” in a customizable directory hierarchy, or *namespace*:
- Several directories can be mounted at the same point – replaces the need for a \$PATH

```
bind /home/shilad/bin /bin
```

```
bind /i386/bin /bin
```

- Copying a local file *foo* to a remote floppy:

```
import lab.pc /a: /n/dos
```

```
cp foo /n/dos/
```

- Piping the output of *foo* on *host1* to *bar* on *host2*

```
cpu -h host1 foo | cpu -h hos2 bar
```



Communication Protocol

- File-based API supports 17 standard I/O file operations
- “Device drivers” (e.g. a telnet server) need only implement the 17 operations.
- A mounting service transparently transmits local file commands to remote systems using the 9P protocol.

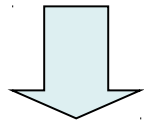


Synchronization

- rendezvous system call:

Thread 1

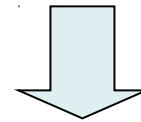
`rendevous("foo", value1)`



value2

Thread 2

`rendezvous("foo", value2)`



value1

both calls block
until they receive
the "foo" tag



Security

- No superuser
- An authentication server maintains a database of authentication keys for all users
- Both client and server exchange encryption challenges and responses so each side is convinced of the others' identity.
- All 9P network communications are secure



Plan 9 Failure

- In 1998, Lucent pulled developers off of Plan 9 to focus on related **Inferno** OS.
- Neither Plan 9 nor Inferno were commercially successful.
- Most experts agree that Plan 9 was technically superior to UNIX but...
- Most experts agree that Plan 9 was not superior *enough* to convert users of UNIX



Plan 9 Success

- Plan 9 open-sourced in 2000
- Dedicated developer and user base remain today
- Google Summer of Code Project
- Plan 9 influenced Linux / BSD:
 - /proc filesystem
 - clone() interface
 - As of 2.6.14, 9P-based file device drivers in stable Linux Kernel



Essential Plan 9

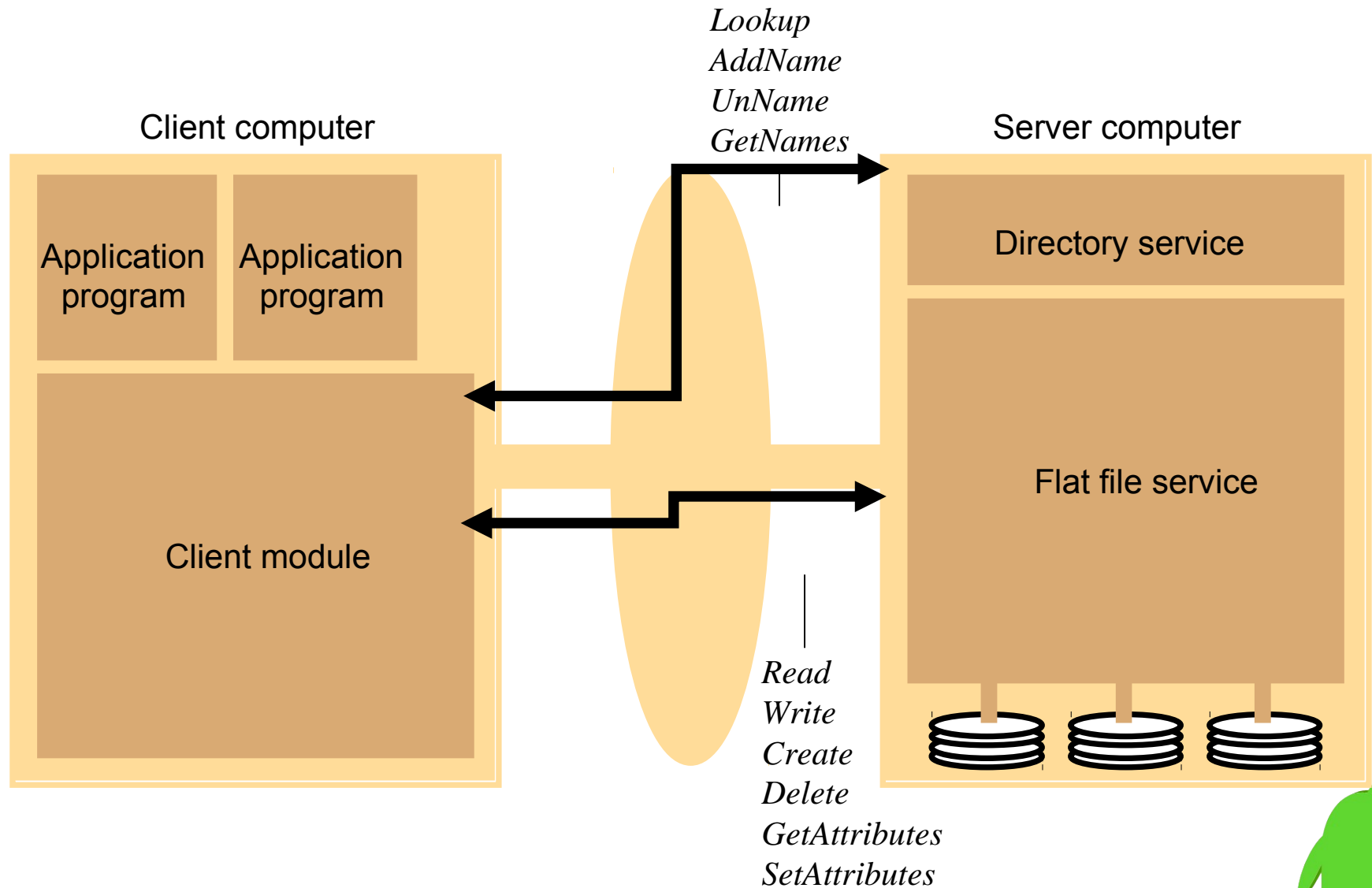


Definitions

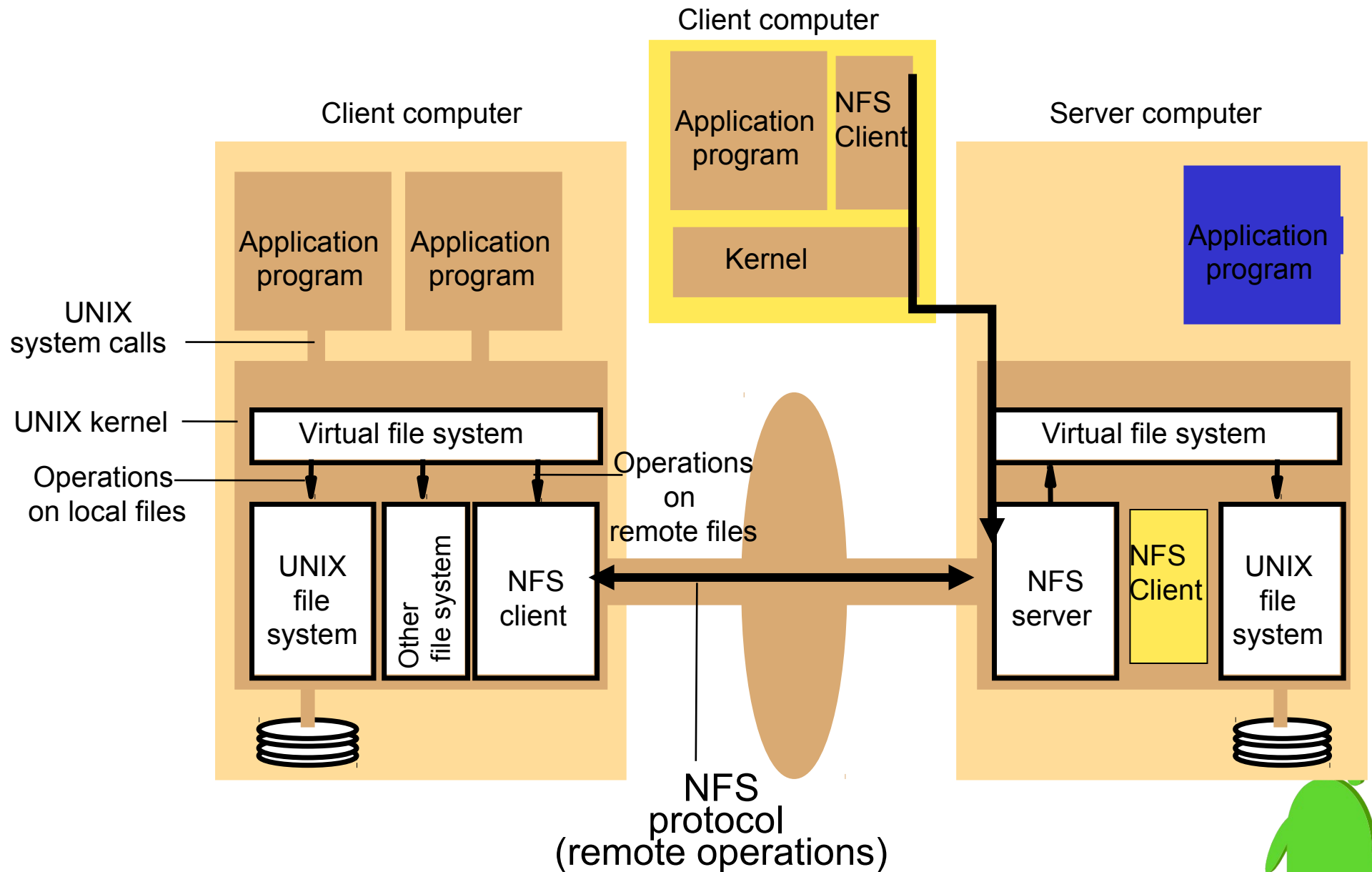
- **8½**
 - The Plan 9 window system, which provides textual I/O and bitmap graphic services to both local and remote client programs by offering a multiplexed file service to those clients. It serves traditional UNIX files like `/dev/tty` as well as more unusual ones that provide access to the mouse and the raw screen. Bitmap graphics operations are provided by serving a file called `/dev/bitblt` that interprets client messages to perform raster operations.
- **Rc**
 - A command interpreter for Plan 9 that provides similar facilities to UNIX's Bourne shell, with some small additions and less idiosyncratic syntax.
- **9P Protocol**
 - The Plan 9 file system protocol. It is structured as a set of transactions that send a request from a client to a (local or remote) server and return the result.
- **IL Protocol**
 - IL is a custom implemented network protocol to transport the remote procedure call messages 9P.



File service architecture



NFS architecture



NFS server operations (simplified)

fh = file handle:

Filesystem identifier	i-node number	i-node generation
-----------------------	---------------	-------------------

```
read(fh, offset, count) -> attr, data
write(fh, offset, count, data) -> attr
create(dirfh, name, attr) -> newfh, attr
remove(dirfh, name) status
getattr(fh) -> attr
setattr(fh, attr) -> attr
lookup(dirfh, name) -> fh, attr
rename(dirfh, name, todirfh, toname)
link(newdirfh, newname, dirfh, name)
readdir(dirfh, cookie, count) -> entries
symlink(newdirfh, newname, string) -> status
readlink(fh) -> string
mkdir(dirfh, name, attr) -> newfh, attr
rmdir(dirfh, name) -> status
statfs(fh) -> fsstats
```

Model flat file service

Read(FileId, i, n) -> Data

Write(FileId, i, Data)

Create() -> FileId

Delete(FileId)

GetAttributes(FileId) -> Attr

SetAttributes(FileId, Attr)

Model directory service

Lookup(Dir, Name) -> FileId

AddName(Dir, Name, File)

UnName(Dir, Name)

GetNames(Dir, Pattern)

->NameSeq



NFS server operations (simplified)

fh = file handle:

Filesystem identifier	i-node number	i-node generation
-----------------------	---------------	-------------------

```
read(fh, offset, count) -> attr, data
write(fh, offset, count, data) -> attr
create(dirfh, name, attr) -> newfh, attr
remove(dirfh, name) status
getattr(fh) -> attr
setattr(fh, attr) -> attr
lookup(dirfh, name) -> fh, attr
rename(dirfh, name, todirfh, toname)
link(newdirfh, newname, dirfh, name)
readdir(dirfh, cookie, count) -> entries
symlink(newdirfh, newname, string) -> status
readlink(fh) -> string
mkdir(dirfh, name, attr) -> newfh, attr
rmdir(dirfh, name) -> status
statfs(fh) -> fsstats
```

Model flat file service

Read(FileId, i, n) -> Data

Write(FileId, i, Data)

Create() -> FileId

Delete(FileId)

GetAttributes(FileId) -> Attr

SetAttributes(FileId, Attr)

Model directory service

Lookup(Dir, Name) -> FileId

AddName(Dir, Name, File)

UnName(Dir, Name)

GetNames(Dir, Pattern)

->NameSeq

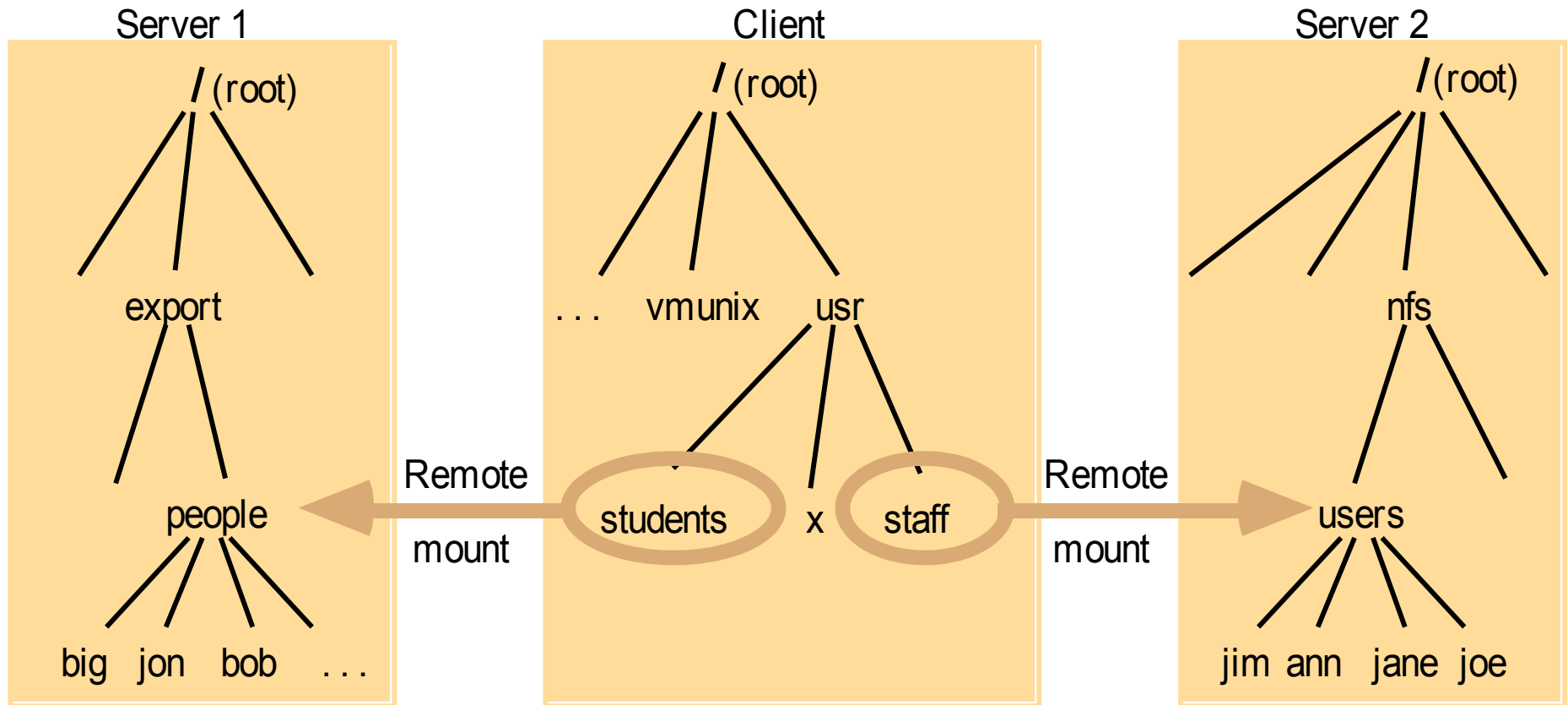


NFS access control and authentication

- Stateless server
 - the user's identity and access rights must be checked by the server on each request.
 - In the local file system they are checked only on *open()*
- Every client request is accompanied by the userID and groupID
 - inserted by the RPC system
- Server is exposed to imposter attacks unless the userID and groupID are protected by encryption
 - Kerberos has been integrated with NFS to provide a stronger and more comprehensive security solution



Local and remote file systems accessible on an NFS client

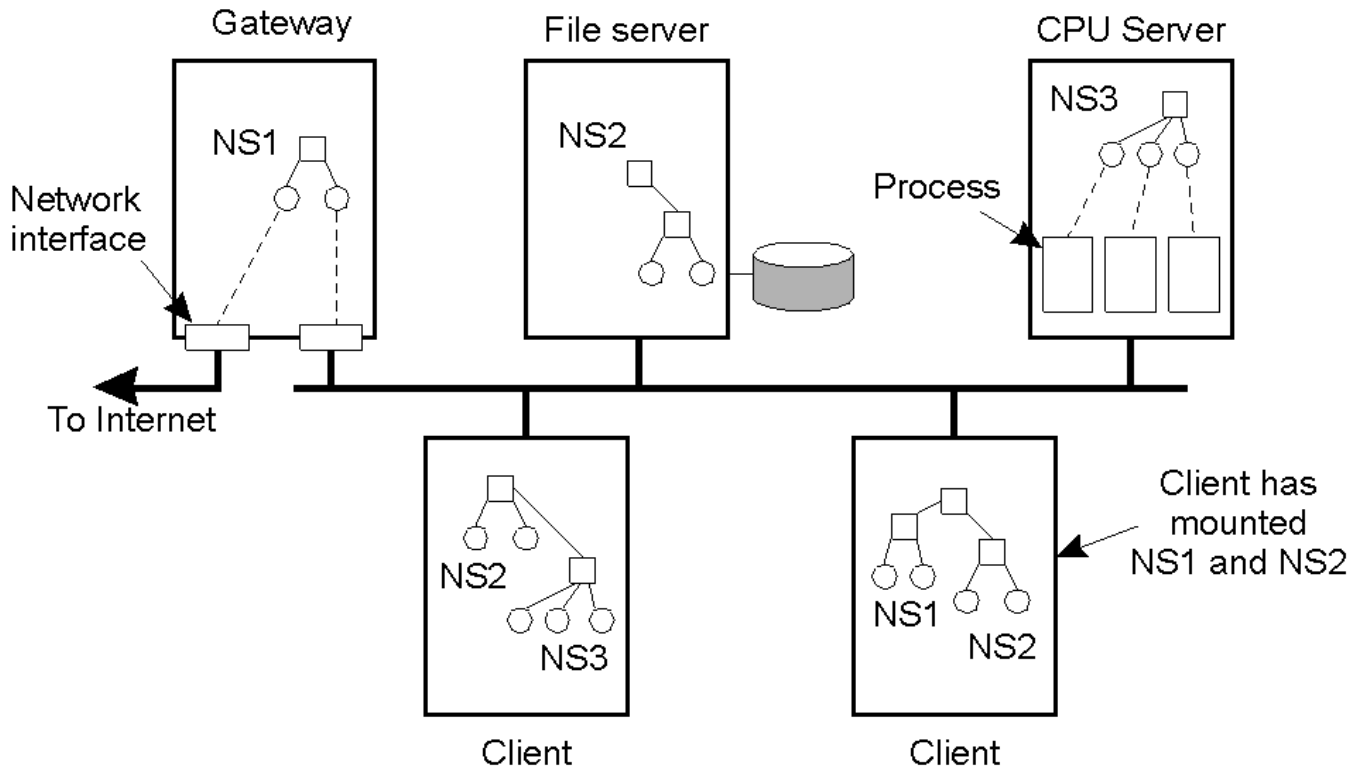


Note: The file system mounted at `/usr/students` in the client is actually the sub-tree located at `/export/people` in Server 1; the file system mounted at `/usr/staff` in the client is actually the sub-tree located at `/nfs/users` in Server 2.



Plan 9: Resource Unified to Files

- All resources in Plan 9 look like file systems.
- File Oriented access
 - Hierarchical name tree
 - Accessible by name
 - Access contents by read and write calls



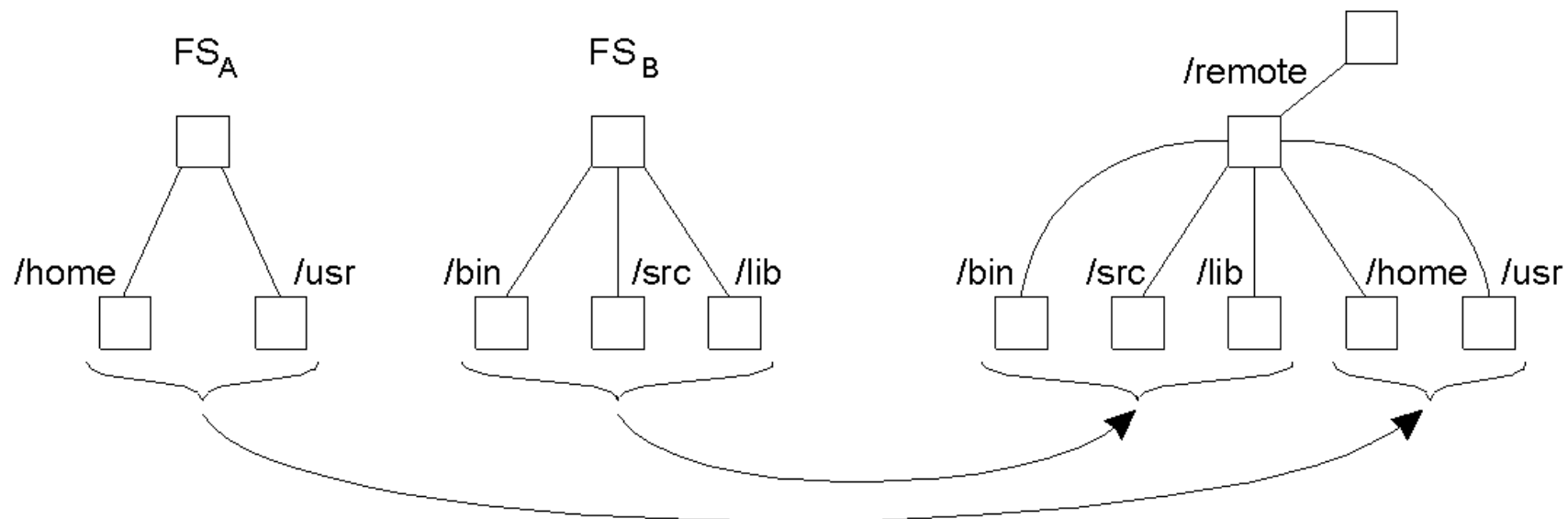
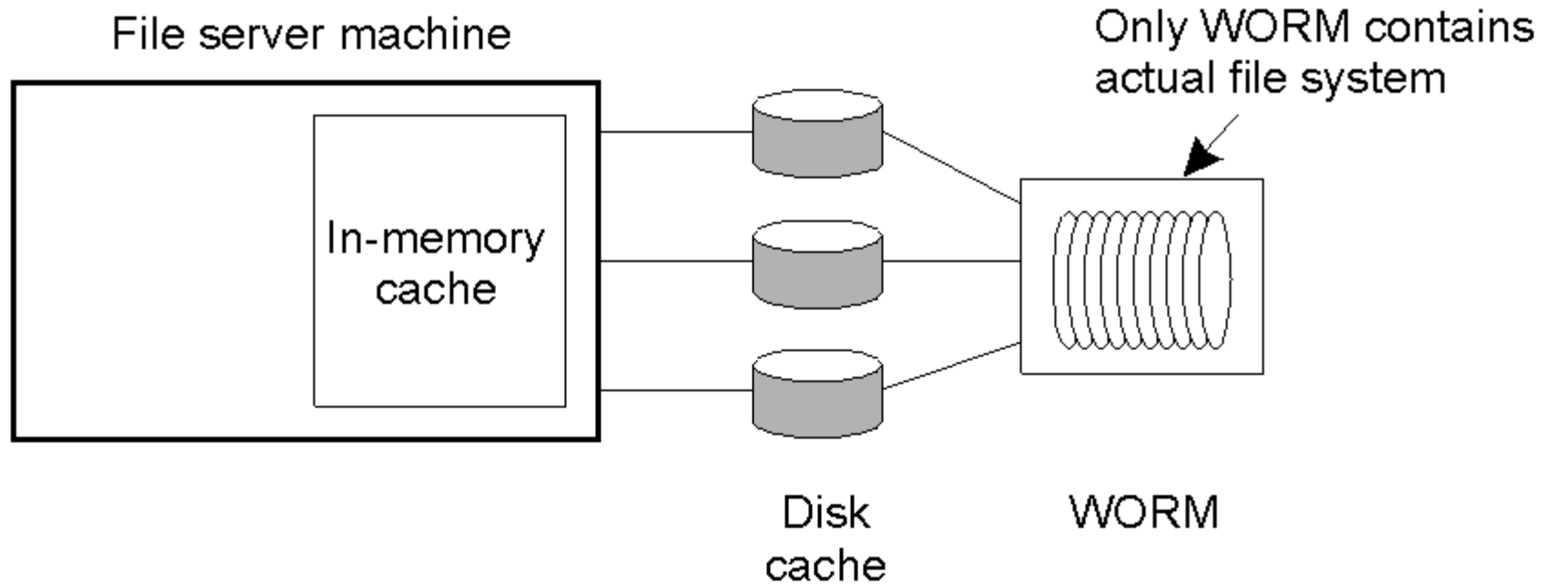
Plan 9 Communications

File	Description
ctl	Used to write protocol-specific control commands
data	Used to read and write data
listen	Used to accept incoming connection setup requests
local	Provides information on the caller's side of the connection
remote	Provides information on the other side of the connection
status	Provides diagnostic information on the current status of the connection

Files associated with a single TCP connection in Plan 9.

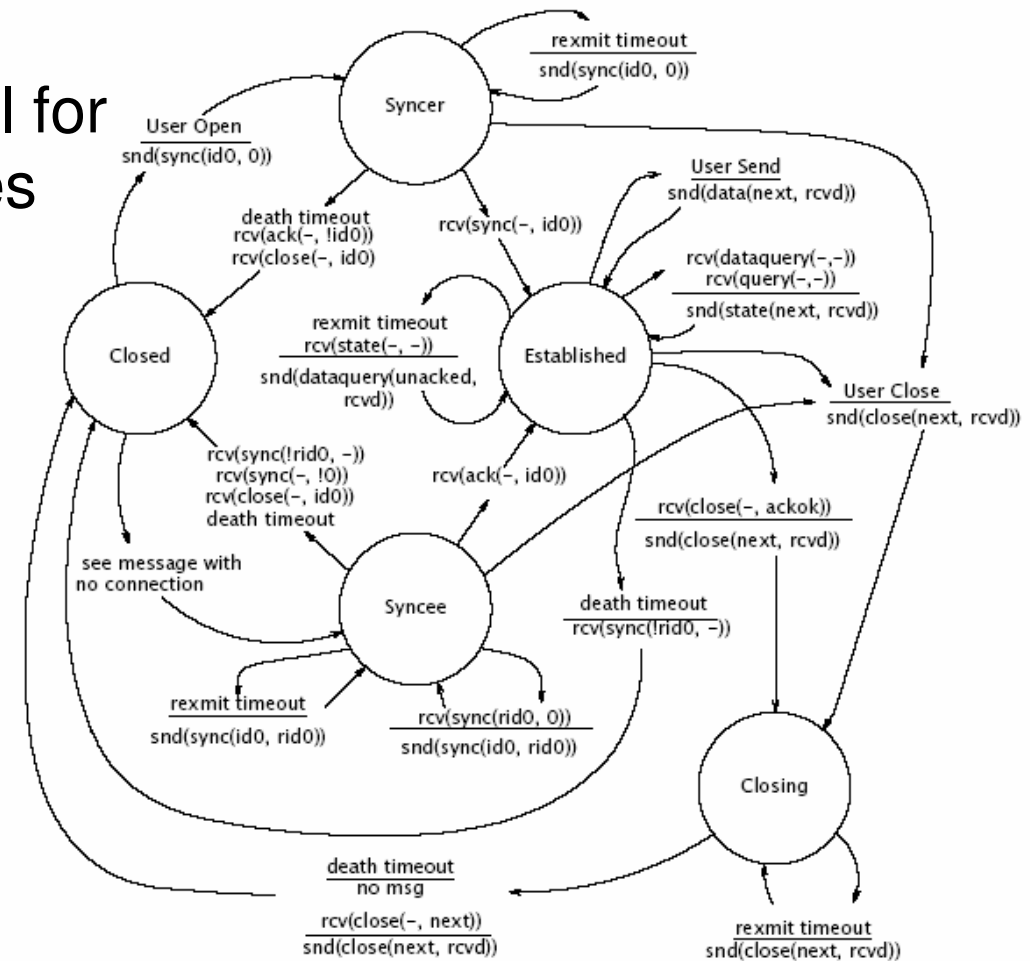


File server & Naming: union directory



9P Protocol

- Standard Protocol for accessing resources
- Bell labs implemented IL Protocol for network transport of 9P messages



ackok

any sequence number between id0 and next inclusive

!x

any value except x

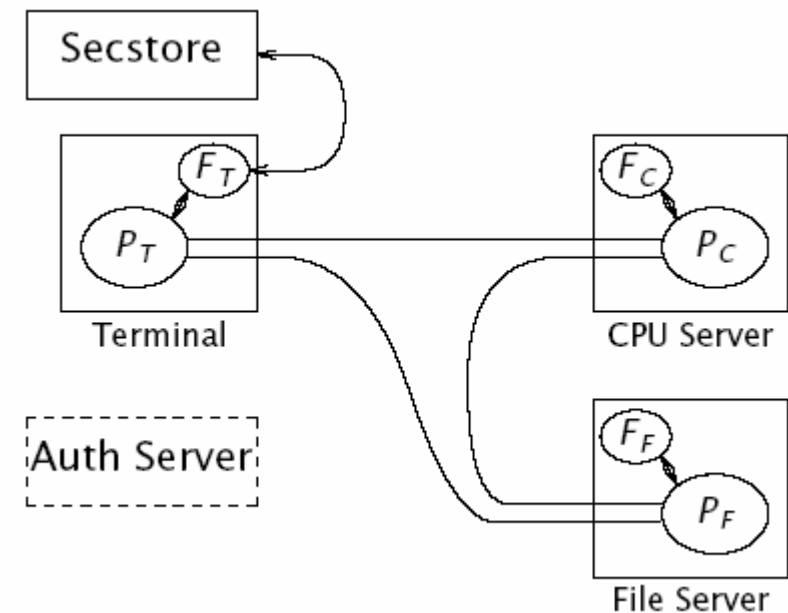
-

any value

Figure 1 – IL State Transitions

Security

- Factotum is the central component of the security architecture. Factotum securely holds a copy of the user's keys and negotiates authentication protocols, on behalf of the user, with secure services around the network
 - Each box is a (typically) separate machine
 - Ellipse is a process. / F_x are factotum processes
 - P_x are the pieces and proxies of a distributed program.



The authentication server is one of several repositories for users security information that factotum processes consult as required.

Secstore is a shared resource for storing private information such as keys; factotum consults it for the user during bootstrap

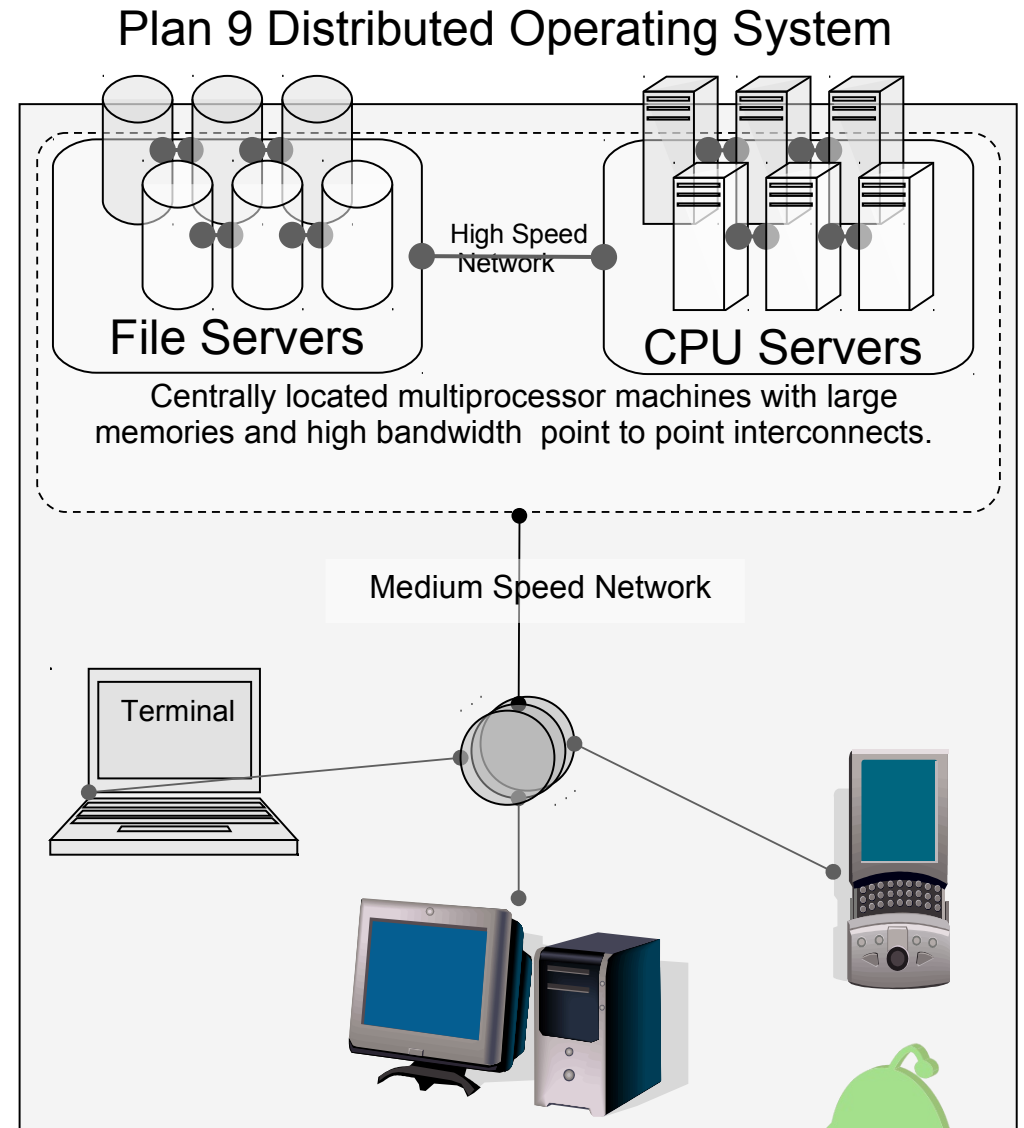
Capability for Parallel Programming

- Kernel provides a simple process model and a few carefully designed system calls for synchronization and sharing.
- Parallel programming language called Alef supports concurrent programming.
- Although it is possible to write parallel programs in C, Alef is the parallel language of choice.
- Alef uses a system call called rendezvous to provides a way for processes to synchronize.



Structure

- Main Parts
 - File servers
 - CPU servers
 - Terminals (user access points).
- Typically centrally located file servers and CPU servers



Using Plan 9

- Example: Echo Server
- This Code implements a typical TCP listener.
- It announces itself
- listens for connections, and for each a new process for each.
- The new process echoes data received on the connection until the remote end closes it.
- The "*" in the symbolic name means the announcement is valid for any addresses bound to the machine the program is run on.

```
int
echo_server(void)
{
    int dfd, lcfd;
    char aaddr[40], laddr[40];
    int n;
    char buf[256];

    afd = announce("tcp!*!echo", aaddr);
    if(afd < 0)
        return -1;

    for(;;){
        /* listen for a call */
        lcfd = listen(aaddr, laddr);
        if(lcfd < 0)
            return -1;

        /* fork a process to echo */
        switch(fork()){
            case 0:
                /* accept the call and open the data file */
                dfd = accept(lcfd, laddr);
                if(dfd < 0)
                    return -1;

                /* echo until EOF */
                while((n = read(dfd, buf, sizeof(buf))) > 0)
                    write(dfd, buf, n);
                exits(0);
            case -1:
                perror("forking");
            default:
                close(lcfd);
                break;
        }
    }
}
```


Using Plan 9

■ Example: Echo Server

1. Announce()

2. Listen()

3. Accept() || Reject()

4. Process()....

5. Close()...

```
int
echo_server(void)
{
    int dfd, lcfd;
    char adir[40], ldir[40];
    int n;
    char buf[256];

    afd = announce("tcp!*!echo", adir);
    if(afd < 0)
        return -1;

    for(;;){
        /* listen for a call */
        lcfd = listen(adir, ldir);
        if(lcfd < 0)
            return -1;

        /* fork a process to echo */
        switch(fork()){
            case 0:
                /* accept the call and open the data file */
                dfd = accept(lcfd, ldir);
                if(dfd < 0)
                    return -1;

                /* echo until EOF */
                while((n = read(dfd, buf, sizeof(buf))) > 0)
                    write(dfd, buf, n);
                exits(0);
            case -1:
                perror("forking");
            default:
                close(lcfd);
                break;
        }
    }
}
```

Plan 9: Using Plan 9

- Example: Echo Server

- 1. Announce()

- Returns open file descriptor for the ctl file of a connection and fills dir with the path of the protocol directory for the announcement.

- `int announce(char *addr, char *dir)`

Addr is the symbolic name/address announced; if it does not contain a service, the announcement is for all services not explicitly announced.

```
int
echo_server(void)
{
    int dfd, lcfd;
    char aaddr[40], ldir[40];
    int n;
    char buf[256];

    afd = announce("tcp!!echo", aaddr);
    if(afd < 0)
        return -1;

    for(;;){
        /* listen for a call */
        lcfd = listen(aaddr, ldir);
        if(lcfd < 0)
            return -1;

        /* fork a process to echo */
        switch(fork()){
            case 0:
                /* accept the call and open the data file */
                dfd = accept(lcfd, ldir);
                if(dfd < 0)
                    return -1;

                /* echo until EOF */
                while((n = read(dfd, buf, sizeof(buf))) > 0)
                    write(dfd, buf, n);
                exits(0);
            case -1:
                perror("forking");
            default:
                close(lcfd);
                break;
        }
    }
}
```

Plan 9: Using Plan 9

- Example: Echo Server
- 2. Listen()
 - Listen returns an open file descriptor for the ctl file and fills ldir with the path of the protocol directory for the received connection.
 - It is passed dir from the announcement.
 - `int listen(char *dir, char *ldir)`

```
int
echo_server(void)
{
    int dfd, lcfd;
    char aadir[40], ldir[40];
    int n;
    char buf[256];

    afd = announce("tcp!!echo", aadir);
    if(afd < 0)
        return -1;

    for(;;){
        /* listen for a call */
        lcfd = listen(aadir, ldir);
        if(lcfd < 0)
            return -1;

        /* fork a process to echo */
        switch(fork()){
        case 0:
            /* accept the call and open the data file */
            dfd = accept(lcfd, ldir);
            if(dfd < 0)
                return -1;

            /* echo until EOF */
            while((n = read(dfd, buf, sizeof(buf))) > 0)
                write(dfd, buf, n);
            exits(0);
        case -1:
            perror("forking");
        default:
            close(lcfd);
            break;
        }
    }
}
```

Plan 9: Using Plan 9

- Example: Echo Server
- 3. Accept() and Reject()
 - Accept and reject are called with the control file descriptor and ldir returned by listen.
 - Some networks such as Datakit accept a reason for a rejection; networks such as IP ignore the third argument.
- `int accept(int ctl, char *ldir)`
- `int reject(int ctl, char *ldir, char *reason)`

```
int
echo_server(void)
{
    int dfd, lcfd;
    char adir[40], ldir[40];
    int n;
    char buf[256];

    afd = announce("tcp!!echo", adir);
    if(afd < 0)
        return -1;

    for(;;){
        /* listen for a call */
        lcfd = listen(adir, ldir);
        if(lcfd < 0)
            return -1;

        /* fork a process to echo */
        switch(fork()){
            case 0:
                /* accept the call and open the data file */
                dfd = accept(lcfd, ldir);
                if(dfd < 0)
                    return -1;

                /* echo until EOF */
                while((n = read(dfd, buf, sizeof(buf))) > 0)
                    write(dfd, buf, n);
                exits(0);
            case -1:
                perror("forking");
            default:
                close(lcfd);
                break;
        }
    }
}
```

Using Plan 9

■ Example: Echo Server

1. Announce()

2. Listen()

3. Accept() || Reject()

4. Process()....

5. Close()...

```
int
echo_server(void)
{
    int dfd, lcfd;
    char adir[40], ldir[40];
    int n;
    char buf[256];

    afd = announce("tcp!*!echo", adir);
    if(afd < 0)
        return -1;

    for(;;){
        /* listen for a call */
        lcfd = listen(adir, ldir);
        if(lcfd < 0)
            return -1;

        /* fork a process to echo */
        switch(fork()){
            case 0:
                /* accept the call and open the data file */
                dfd = accept(lcfd, ldir);
                if(dfd < 0)
                    return -1;

                /* echo until EOF */
                while((n = read(dfd, buf, sizeof(buf))) > 0)
                    write(dfd, buf, n);
                exits(0);
            case -1:
                perror("forking");
            default:
                close(lcfd);
                break;
        }
    }
}
```

Significance

- Fulfills definition of Distributed System
- 9P: Uses message passing to coordinate actions
- Provides for the major design considerations as outlined
- *Heterogeneity, Openness, Security, Scalability, Failure handling, Concurrency, Transparency*



Renaissance



Reference

- Unix and Beyond: Themes in Operating Systems Research, Dennis M. Ritchie, Bell Laboratories, Alcatel Lucent (2007)
- Plan 9 from Bell Labs: Real World Distributed Operating System Case Study, Bryan Kinney, Computer Science and Engineering, Christopher Newport University (2005)





<http://0xlab.org>