15-412

P9 / 9P
Nov. 13, 2019

Dave Eckhardt

Outline

The Boot Story

- Unix
- File server
- CPU server
- Terminal

9P in a hurry

P9 in a hurry

- Skipping the unimportant parts
  - VM
  - Scheduling
  - Name spaces

The Plan 9 Approach

"Build a UNIX out of little systems"

- ...not "a system out of little Unixes"

Compatibility of essence with Unix

- Not real portability

Take the good things

- Tree-structured file system
- "Everything is a file" model

Toss the rest (ttys, *signals!*)

Design Principles

“Everything is a file”

- Standard *naming system* for all resources (pathnames)

“Remote access” is the common case

- Standard *resource access protocol*, 9P
- Used to access any file-like thing, remote or local

Personal namespaces

- Naming *conventions* keep it sane

System Architecture

Reliable machine-room *file servers*

- Plan 9's eternal versioned file system

Shared-memory multiprocessor *cycle 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
  - (most of it)

Boot - Unix

"Somehow" a kernel starts running

Last steps before user-mode

- Mount "the root file system"
  - A string in the binary/in the environment
    - » ufs:/dev/ad0s3a
    - » ufs:/dev/md0
    - » cd9660:/dev/acd0a
- Run a program from that fs as "root" (uid 0): /sbin/init

The "superuser" (uid 0) controls all, can become any

Boot – Plan 9 File Server

"Somehow" a kernel starts running

- With some configuration strings somewhere in (NV)?RAM

Kernel contains a "RAM disk" file system

- Specified in config file, e.g., /sys/src/9/pc/pccpuf
  - /boot (an executable file, boot(8))
  - /386/bin/ip/ipconfig
  - /386/bin/auth/factotum
  - /386/bin/disk/kfs
  - /386/bin/fossil/fossil
  - /386/bin/venti/venti
  - A few blank directories, e.g., /mnt and /dev

Last step before user-mode - exec("/boot")

Boot – Plan 9 CPU Server

"Somehow" a kernel starts running

- With some configuration strings somewhere in (NV)?RAM

Kernel contains a "RAM disk" file system

- Specified in config file, e.g., /sys/src/9/pc/pccpuf
  - /boot (an executable file, boot(8))
  - /386/bin/ip/ipconfig
  - /386/bin/auth/factotum
  - A few blank directories, e.g., /mnt and /dev

Last step before user-mode – exec("/boot")

- Launch factotum
- Configure network stack
- Connect to file server, authenticate, mount("/")

Boot – Plan 9 Terminal

"Somehow" a kernel starts running

- With some configuration strings somewhere in (NV)?RAM

Kernel contains a "RAM disk" file system

- Specified in config file, e.g., /sys/src/9/pc/pccpuf
  - /boot (an executable file, boot(8))
  - /386/bin/ip/ipconfig
  - /386/bin/auth/factotum
  - A few blank directories, e.g., /mnt and /dev

Last step before user-mode – exec("/boot")

- Launch factotum
- Configure network stack
- Connect to file server, authenticate, mount("/")

Differences?

Versus Unix

- There is no “super user”

Differences?

Versus Unix

- There is no “super user”
- There is no uid 0

Differences?

Versus Unix

- There is no "super user"
- There is no uid 0
- There are no uids

Differences?

Versus Unix

- There is no “super user”
- There is no uid 0
- There are no uids
  - It's just hopeless to try to pass numeric user id's from one administrative domain to another
    - » The “AFS ls -l” problem
  - The best you can do is to express user@domain
    - » The “AFS fs la” ~solution

Differences?

There is no "super user"

Where does authority come from?

- Control of any machine controls its resources
  - This is real life!
- When you mount files from a file server, you trust that server
  - Especially if you execute binaries from it
- In some sense each file server is a "user"
  - (Kerberos uses the term "principal" for "user or server")
  - Users prove identity to file servers to get and put files
  - File servers prove identity to users during login
- If you control a machine, you own its disks, etc. ...

Differences – Plan 9 Kernels

Somewhere there must be an authentication server

- Otherwise all those “authenticate & attach” steps fail
- (Inferno has a distributed certificate-based approach)
- Authentication server may be a “very small file server” which exports no files

Differences – Plan 9 Kernels

CPU server vs. File server

- CPU server lets people “log in” and run programs
- If there's a disk, it contains a kernel and maybe swap space
- So the “RAM disk” doesn't contain file-server code

Differences – Plan 9 Kernels

CPU server vs. Terminal

- Both need enough programs to configure network and authenticate to file server
- Terminal devotes more RAM to screen image buffers in kernel (/dev/draw)
- Terminal expects to prompt "the user" for authentication information
- CPU server needs to mount file server without a person typing a password
  - So a "username" and password must be stored on disk
    - » On good platforms (non-PC), store in NVRAM

Differences – Unix vs. P9

Where is “the file system”?

- Unix: on the “root partition” of “the FreeBSD slice”
- Plan 9
  - There is no “the file system”
  - Each kernel exports a handful of file systems
    - /dev/pci
    - /proc
    - /net/tcp
    - /dev/sdC0
  - Kernel file systems are “small” (usually < 1,000 files)
    - Frequently 2, e.g., /dev/eia0, /dev/eia0ctl
  - Kernel file systems are “dynamic” (reset on reboot)

Differences – Unix vs. P9

But where are the files???

- In a special (non-volatile) file system
  - Eckhardt terminology: “file store”
- Originally served by a “Ken fs”
  - Ken Thompson's fileserver-only kernel
  - A Plan 9 kernel running a handful of kernel processes
  - No user space
- These days served by a user process “on some machine”
  - kfs (!= “Ken fs”, but directly inspired)
  - cwfs is Ken fs, ported to user mode, simualting WORMs
  - fossil (like Ken fs/cwfs, does snapshots and history – in Venti)
- Isn't that weird?
  - Maybe (but see: AFS)

Differences – Unix vs. P9

How do I access the files?

- Unix
  - Files in “the file system” are accessed via the kernel
    - » Maybe there are multiple file systems, with different architectures
    - » Maybe each time one is added parts of “the VFS layer” shift around
  - Files “somewhere else” are also accessed via the kernel
    - » AFS files: via the AFS-cache-manager kernel module
    - » NFS files: via the NFS-thingie kernel module
    - » ...
- Plan 9
  - All file system calls translated to one object access protocol
  - Each 9P request is sent to the appropriate server

9P Overview

"fid"

- 4-byte number, chosen by client
  - "When I say 51, I will refer to ..."

"qid"

- 13-byte number, chosen by server
  - 1-byte type (directory, append-only, do-not-archive, ...)
  - 8-byte "path"
    - » Unique id # of a "particular file"
    - » "rm foo ; touch foo" ⇒ hopefully two different qid paths
  - 4-byte "version"
    - » Increments when file changes

9P – Attach, Authenticate

authenticate(username, attachpoint, fid)

- T: I, davide, wish to authenticate to your “main” file system by doing I/O to fid 13
- R: Go right ahead. The qid of fid 13 is ...

attach(username, attachpoint, afid)

- T: I, davide, who have already proven my identity by manipulating the authentication file 13, wish to attach to your “main” file system. I will refer to the root of the file system by fid 0.
- R: Ok. The qid of fid 0 is ...

9P – From Here to There

walk(fid, newfid, nnames, names[])

- T: Starting from the directory specified by fid 0, please walk down names[0], names[1], etc., and call the result fid 1.
- R: Ok. The qid of names[0] is ...; the qid of names[0]/names[1] is ... ...

clunk(fid)

- T: I will no longer refer to fid 77.
- R: Thanks!

9P – I/O (At Last!)

read(fid, offset, count)

- T: I would like to read the first 4096 bytes of fid 33.
- R: Ok. Bytes follow: ...

write(fid, offset, count, data[])

stat(fid), wstat(fid)

remove(fid)

9P – Permissions

Each file has “Unix permission bits”

- {owner, group, world} X {read, write, execute}

Each client has a username

- The username it proved it represented via auth()

Each client username is a member of some groups

- As determined by the file server (Hmm....)

[Discuss]

P9 – What Does open() Mean?

open() means

- Convert my parameters to a Twalk request
- Write the fid down somewhere

read() means

- Convert my parameters to a Tread request
  - Recalling the fid and my current file offset
- Send the request, put me to sleep until the response comes back, ...

seek() means

- Adjust my current file offset (don't convert into anything)

9P – Special Case for Kernel FS's

What logically happens

- Kernel converts read() to Tread, sent to /dev/cons "file server"
- /dev/cons "file server" converts Tread to cons_read()
- Result converted to Rread, sent back to client
- Kernel converts Rread to read() results

What actually happens

- Kernel converts read() to cons_read()
- cons_read() has more control over the process than a remote file server
  - "echo reboot > /dev/reboot"

9P – Being a Server

Libraries for user space

- 9p(2)
  - “server RPC dispatch skeleton”
  - Conceptually, you write “the methods” and it does the “housekeeping”
- 9pfid(2)
  - fid management, used mostly by 9p(2)
- 9pfile(2)
  - Implements most of a “small” “in-RAM” file system
    - » Directories, I/O
    - » You provide the actual bits
- Reading source can be illuminating even if the library isn't for you

9P – Being a Server

Kernel files

- Many device drivers export a small number of files
  - For many it's a small *constant* number
- Often “all” the files are in one directory
- Standard boilerplate code for implementing “the directory” and the (not-really-extant) “fid ⇒ qid” mapping
- Often file I/O is dispatched by low-order bits of qid path
  - In other words, qid path 0 is “ctl”, 1 is “raw”, ...

What Does “mount” Mean?

Unix

- Activate some kernel module, pass it a string
- It will present “that resource” to everybody who references /some/pathname

Plan 9

- Here is a file descriptor connected to my key manager
  - Please connect my key manager up to an afid on the server at the other end of this other file descriptor
  - Please obtain a fid for “/” of that file server's “foo” tree
- From now on, whenver I refer to /some/path, turn all my system calls into 9P requests
- Nobody else is affected at all

Differences – Unix vs. P9

Do I *need* mount()?

- Unix
  - Only the superuser can do anything so dangerous!!!
  - Certainly you can't talk to “the file system” directly, since it's safely somewhere on “the disk”.
  - You can access whatever the superuser said you can access – and you should be grateful!
- Plan 9
  - No. Anything you can access through mount you could also access by sending 9P requests on the file descriptor yourself.

Differences – Unix vs. P9

"Superuser"?

- Unix
  - The mystical uid 0 can do "everything"
  - On which machines? Hmmm...
- Plan 9
  - The "host owner" of each host owns the resources of that host
  - The host owner owns the CPU, so can manipulate /proc and kill processes
  - The host owner owns the disks, so can run file-server processes which open disk partitions
  - The host owner owns the whole machine, so can cause any process to take on any username string

Left Out

The whole authentication thing

- There is an “auth server” much like a Kerberos KDC
- There is an “authentication file system” for each user “logged in to the system” (system meaning all the nodes)
- All authentication code (client, server) is in the auth server or in the per-user authentication file system (called factotum)
- A story for another day

Summary

The Boot Story

9P in a hurry

P9 in a hurry