A read-only shared NFS root filesystem for a group of diskless machines

Ignatios Souvatzis

Institut für Informatik

RHEINISCHE FRIEDRICH-WILHELMS-UNIVERSITÄT

ignatios@cs.uni-bonn.de

http://theory.cs.uni-bonn.de/~ignatios/

Introduction

- Manual managment of 3 diskless workstations is fine
- Manual managment of 10 diskless workstations is possible, but tedious.
- Manual managment of hundreds is close to impossible

My task a few years ago

to set up 10 diskless, headless DNARD machines as computing nodes for a PVM[2] practical course.

My idea

let them share a common root file system to ease administration. While we're here, also make it read-only, for increased system security.

Structure

- Introduction
- Environment
- Why a shared root?
- Problems
- Methods
- Implementation details
- Why no kernel-embedded root file system?
- Further reading

Color codes in the slides

- dangerous
- fine
- changed between NetBSD-1.5 (paper) and -1.6
- normal
- not yet implemented

System environment

Boot- and fileserver

hardware	Sun UltraSparc-10
OS	Solaris 2.6 (now Solaris 8)
location	restricted access

Student home directories are on a different machine elsewhere.

Client machines

hardware	Digital Network Appliance Reference Design (“shark”);
CPU	233 MHz StrongARM-110
memory	64 MBytes
OS	NetBSD-1.4 (now NetBSD-1.6)
location	restricted access

Users

- users are about a dozen students, changing twice a year
- should be able to log in from terminal room or home
- we don’t know what we can expect from them

Why a shared read-only root file system?

- Administration
- Security

Administration

- easy administration: change configuration once on server, and maybe run startup script on clients or reboot them
- save disk space: as of NetBSD-1.5 or -1.6, 20 MB per client (counting */bin, /sbin, /etc*)

System security

- The clients need root access to their root area - but root access to the server should be restricted as far as possible.
- Should somebody break into a client, we can change software and configuration to prevent further break-ins knowing that the files on the server weren't changed.
- *But this doesn't prevent access to NFS accessed user data!*

Problems encountered...

- while booting
- during operation
- during shutdown

... while booting

- client has to learn its network address and name
- client has to learn the name of its NFS swap file
- client has to learn the name of per-client file systems on the server

Obvious solution

The DNARD firmware uses DHCP to boot (via IPv4), so...

- use DHCP to learn name, address, root filesystem
- derive swap file name etc. from client name
- use IPv6 “stateless address autoconfiguration” (or maybe DHCPv6)

... during operation

Files which are written to the root file system on traditional Unix, or are tied to a single machine:

- System log files;
- temporary files in */tmp* (don't survive reboot)
- temporary files in */var/tmp* (may survive reboot)
- **.pid* - files, written (at least) during startup of server processes

- Sockets (*/dev/log, /dev/printer, ...*), created by server processes as communication ports for their clients
- Device nodes (of terminals, mice, keyboards, audio) change owner at each login/logout and access times on each I/O operation
- ssh host key
- package system database
- */etc/motd* update at late boot time

... during shutdown

A shared */etc/nologin* prevents login to the clients still running, while a shutdown is in progress on a client!

Methods

- Some problems can be ignored.
- Some directories are changed into MFS file systems.
- Some directories are changed to per-client areas on the server.
- Some software is told to use other places
- Where do we place the needed startup script code?

Some problems aren’t - just ignore them

- all clients are equal - use a single ssh host key
- tell syslog to send log messages to our server
- DHCP was already mentioned - nothing special to setup
- we already use *mfs /tmp*

Sockets and **.pid* - files

Those are on */var/run* in NetBSD. Either

- use a per-client */var* file system (we do this anyway) or
- mount a small memory file system (*mfs*, pageable[a]) on */var/run*.
  - add to */etc/fstab*
  - mark */var/run* as “to be mounted early”

[a] manual pages: mount_mfs(8)

Device nodes

- if */sbin/init* can't open */dev/console*,
  - it mounts an *mfs* on */dev* and
  - runs *MAKEDEV*
- In NetBSD-1.5 we had to do this ourselves in the *shroot* - script.
- additional startup time on our hardware: 20 seconds (for both methods)

Swap and /var

- during boot time, the kernel learns its name via DHCP
- The shroot script runs */bin/hostname* to learn the name from the kernel.
- The shroot script constructs the names for per-client */var* and swap.

How to make */var* read-only

Per-client */var* needed for outgoing e-mail, print jobs etc. and */var/tmp*.

Idea: change/replace client software to:

- directly write to the network
- return error if this isn't possible

and put /var/tmp on mfs, too

If we do this, contents are lost at boot time!

Remaining files written to */etc*

- some can be shut off (e.g., */etc/motd* update at boot time)
- but some can't without lots of code change (*/etc/nologin*)

Solution: union-mount a small *mfs* over /etc (also handled in *shroot*).

Where to place the code?

Our setup script has to run before

- in NetBSD-1.5: devices are accessed
- swap is used
- */var* is used

in other Unices...

- Traditional BSD *rc.local* is too late! — hook into the /etc/rc script
- NetBSD-1.4: /etc/netstart.local
- System V-ish init scripts are ordered by some number in the name - beware of changed system scripts!

in NetBSD-1.5 and later

- NetBSD runs startup scripts with explicit dependencies [a].
- We mark the script as
  - *REQUIRE: root*
  - *BEFORE: mountcritlocal*.

[a] manual pages: rc(8),rcorder(8),rc.d(8)

Installing packages

- PKG_DBDIR=/usr/pkg/pkgdb (/etc/profile, /etc/mk.conf)
- give a selected client write access for the time of the installation and revoke afterwards
- all clients can use *pkg_info* to check installed software

It would be better to have cross-installation tools that run on the server.

Implementation

The most important scripts/configuration files are shown:

- */etc/rc.conf* for the clients
- */etc/rc.d/shroot* script for early startup name finding

Finally, the run-time view of the file systems is presented.

rc.conf

```
rc_configured=YES
shroot_pfx_var=server:/BSD/root/var-
shroot_pfx_swap=server:/BSD/swap/

critical_filesystems_local="/var /var/run"}
#in 1.5: critical_filesystems_beforenet

update_motd=NO
rpcbind=YES # nfs
domainname=nis.doma.in # NIS
ypbind=YES
amd=YES amd_dir=/var/amdroot
nfs_client=YES
ip6mode=autohost
```

```
defaultroute=10.10.10.10
sshd=YES
postfix=YES
ntpd=YES
lpd=YES lpd_flags=-s
inetd=YES        # ntalk, (c)fingerd
```

shroot

```
#!/bin/sh
# PROVIDE: shroot
# REQUIRE: root
# BEFORE: mountcritlocal
# shared root setup.
. /etc/rc.subr

name="shroot"
start_cmd="shroot_start"
stop_cmd=":"
```

```
# 1.5: required_files=\
# "/sbin/MAKEDEV /sbin/MAKEDEV.local"


shroot_start () {
  hostname=`/bin/hostname`

  case "$shroot_pfx_var" in
  "") ;;
   *) /sbin/mount -t nfs \
      ${shroot_pfx_var}${hostname} /var ;;
  esac
  case "$shroot_pfx_swap" in
  "") ;;
  *) /sbin/mount -t nfs \
```

```
        ${shroot_pfx_swap}${hostname} /swap\
          && /sbin/swapon /swap ;;
    esac
    /sbin/mount -t mfs -o union \
      -o -i=256 -o -s=512 swap /etc
    /bin/chmod 755 /etc
```

```
# 1.5: /sbin/mount -t mfs -o -i=256 \
#                  -o -s=512 swap /dev
# not needed for 1.6:
# /bin/chmod 755 /dev
# echo -n "creating device nodes...";
# /bin/cp /sbin/MAKEDEV /dev
# /bin/cp /sbin/MAKEDEV.local /dev
# (cd /dev; sh MAKEDEV all)
# echo done.
```

```
# end of shroot():
}

load_rc_config $name
run_rc_command "$1"
```

Run-time view of file systems

```
> mount
server:/BSD/root2/1.5 on / type nfs (read-only)
server:/BSD/root/var-client on /var type nfs
server:/BSD/swap/client on /swap type nfs
mfs:34 on /dev type mfs (asynchronous, local)
mfs:37 on /etc type mfs (asynchronous, local, union)
mfs:1000 on /var/run type mfs (asynchronous, local)
server:/BSD/root/vargames on /var/games type nfs
mfs:1085 on /tmp type mfs (asynchronous, local)
pid1121@client:/home on /home type nfs
studsrv:/opt/export/home/stud/user on \
        /var/amdroot/studsrv/opt/export/home/stud/user \
        type nfs (nodev, nosuid)
server:/export/home/2 on /var/amdroot/server/export/home/2 \
        type nfs (nodev, nosuid)
```

Why no kernel-embedded root file system?

root file system	shared	embedded
root fs RAM usage	on demand	fixed
installation	standard	select to fit
after configuration	run script or reboot	reboot client

References

[1] Home page of the course:

http://theory.cs.uni-bonn.de/infoV/system/parlab/[a]

[2] PVM home page: *http://www.epm.ornl.gov/pvm/*

[3] NetBSD manual pages mount_mfs(8),rc(8),rcorder(8)

[4] NetBSD/shark port page:

http://www.netbsd.org/Ports/shark/

[a] (sorry, currently German language only)