Diskless Architecture

1.0 Introduction

A couple of factors necessitated a modified diskless architecture. The major change was the adoption of SVR4 directory hierarchies. Get a copy of *"Directory Structure Re-organization in IRIX 5.x*" if you are unfamiliar with SVR4 hierarchies. A postscript copy is stored in the RCS tree as ".../irix/cmd/diskless/dir-reorg.ps". A good familiarity with the IRIX directory structure change is pertinent to understanding this architecture document. A second factor was the need to allow sharing across architectural boundaries and make our implementation look similar to other vendors.

It is not the intent of this document to describe the installation process in detail. Instead, pertinent details required to understand the affects on pre-ops, post-ops, & exit-ops are given. A discussion of developer concerns is at the end of the document.

2.0 Diskless operation

Without discussing specifics about installation, this section will detail the bootup sequence. This information is important to know before installation specifics are explained.

2.1 Preliminary details

A few general items must be satisfied before booting can commence.

2.1.1 client

While at the prom prompt, some prom variables must be set before the client can behave as a diskless client. All platforms (R3k & R4k) must set the **diskless** variable to 1. R3k systems must set the additional variable bootfile to reflect where to pick up the kernel. An example might be:

```
setenv bootfile bootp()foo:/var/boot/bar/unix
```

where **foo** is the diskless server and **bar** is the diskless client.

R4k machines (except Crimson) must set two analogous variables, **SystemPartition** & **OSLoader**.

2.1.2 server

Diskless client software must be installed onto the diskless server. Details will be given later.

2.2 bootp & tftp protocols

The bootp protocol is used to find the unix image sitting on the diskless server. A client workstation's prom parses the **bootfile** variable and treats bootp as a pseudo-device driver. It passes the name of the server and the filename to the bootp protocol code which is burned into the prom. Once the server is found, the tftp protocol is used to download the kernel to the client. This is only loading the kernel image across the network; no execution has occurred.

2.3 bootparam protocol

Once the kernel is loaded, it starts running. At this point, there is no root filesystem and no init process. The running client kernel decides it is running diskless due to the **diskless** prom variable which was set earlier. The bootparam protocol is used to query the diskless server about the client's partitions to mount.

Three variables or partitions are gathered with this protocol: root, /sbin, & /swap. The root partition is private; only the client will write to it. The /sbin partition is a read-only partition which contains those files which are shareable across architectures but needed to bring a machine up to single user mode. The /swap partition is just space for swapping.

2.4 finish booting

After all three bootparam partitions are NFS mounted, the client continues booting just as a normal diskfull machine boots. Along the way, it reads the local /etc/fstab file to find out where its /etc partition is located. /usr is mounted read-only; it holds files which are shareable across all architectures but not needed until after single-user mode.

3.0 Installation synopsis

Installation requires running two shell scripts, *client_inst* and *share_inst*. They are both available in the **nfs.sw.diskless_server** subsystem. After both scripts complete, a client's system software will be installed on the server in three trees: the client tree, the share tree, and the swap tree. For a detailed discussion of diskless installation, see "*Diskless Workstation Administration Guide*" (document # 007-0855-030).

share_inst gets executed first; and at some point, the familiar *inst* menu appears. Subsystems are chosen as during a diskfull install. However, one additional subsystem must be installed to complete the diskless environment, **nfs.sw.diskless_client**. This subsystem is only available during diskless installations. Client tree installations, using *client_inst*, are performed against specific share trees. As such, the subsystems available for installation into a client tree are restricted to the subset installed in the share tree. Swap trees only contain swap partitions and are built as a side effect to running *client_inst*.

3.1 Share Trees

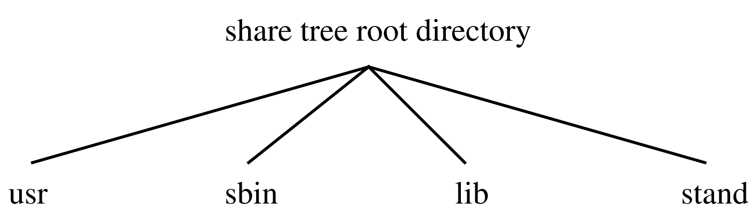
The share tree contains those files which are shareable across architectures. Man pages, binaries, libraries (while everything is MIPS1), and data files are good examples. Because objects are shareable, they are exported from the server in read-only filesystems. In a diskless setup, there should be one share tree per release. For example, if one Indigo R3k and one Indigo R4k want to use IRIX 5.0, they would both use one share tree. If three Indies wanted to run IRIX 5.1.1.1, they would use a different share tree.

3.1.1 Specifics of a share tree

Everything under the following directories is shareable.

- /usr
- /sbin
- /lib
- /stand

Therefore, a share tree looks like:



Although the share tree contains all the binaries on a running system, it is not a complete root. Therefore, using “chroot” in a pre-op, post-op, or exit-op is **not** guaranteed to operate properly since there are only four directories in the share tree root. Even so, some pre/post/exit-ops are performed against the share tree. A /dev directory with a few files has been created in the share tree (not shown above); so well-known, often executed chroots succeed. It was needed because rld must open /dev/zero on start-up. In summary, the commands performed in the operation will determine the success of the pre-op, post-op, or exit-op.

3.1.2 About “share_inst”

share_inst is a shell script which calls *inst* with extra options. It first creates a /dev directory and fills it with required devices like /dev/zero. A symlink is created from “lib -> sbin/lib”. Rather than create another directory analogous to /sbin, this forces *inst* to install /lib library files into the directory /sbin. Architecture specific binaries located under /stand are placed under /sbin/stand and given architecture independent names. When *client_inst* runs, it installs architecture specific symlinks from /stand to this directory. As stated previously, /sbin is a directory for shareable files required to boot to single-user. It takes on additional objects in the diskless environment.

inst is called as such:

```
inst -r$SHARE -s -C -d usr -d sbin -d lib
```

"`-s`" tells *inst* that this is a share tree

"`-C`" tells *inst* to read **/var/boot/.dl_classes** which contains valid diskless architectures

"`-d usr`" tells *inst* to install files beginning with /usr into the share tree.

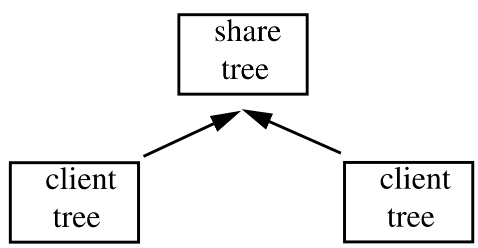
"`-r $SHARE`" tells *inst* where the root of the share tree is located.

Therefore, if an exit-op is attached to a file which begins with /usr and that specific sub-system gets installed, it will get run when *inst* exits during a share tree install. The operation will not get run during a client tree install.

Lastly, *share_inst* NFS exports /sbin and /usr.

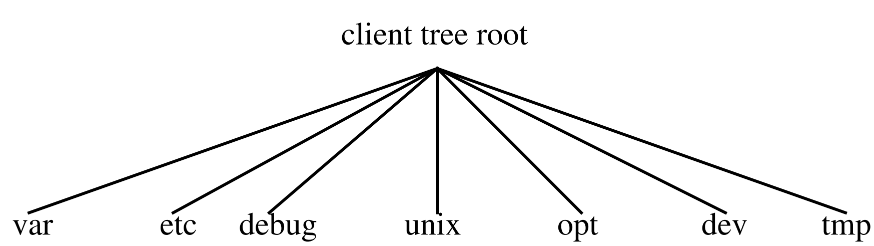
3.2 Client Trees

The client tree contains everything which is not in the share tree. Files beginning with names other than those listed in the share tree (see 3.1.1) are placed in each client tree. The relationship between client trees and share trees is many to one.



3.2.1 Specifics about client trees

A client tree might look like:



There is no way a client tree can be made into a viable root filesystem. There are no binaries in the tree, not even a shell. In the above picture, there is no /bin directory! Therefore, any pre-ops, post-ops, or exit-ops using "chroot" will fail. Shell scripts should be re-written to use full pathnames from the miniroot root.

3.2.2 About "client_inst"

client_inst is a shell script which invokes *inst*. It creates the client's /etc/fstab file, NFS exports the client's root and swap partitions, and enters a line into the server's /etc/bootparams file. The bootparam entry contains the client's root, sbin, and swap partitions which are mounted when the kernel starts.

The script then calls *inst* as such:

```
inst -r$CL_ROOT -S $SHARE -m$MACH ...
```

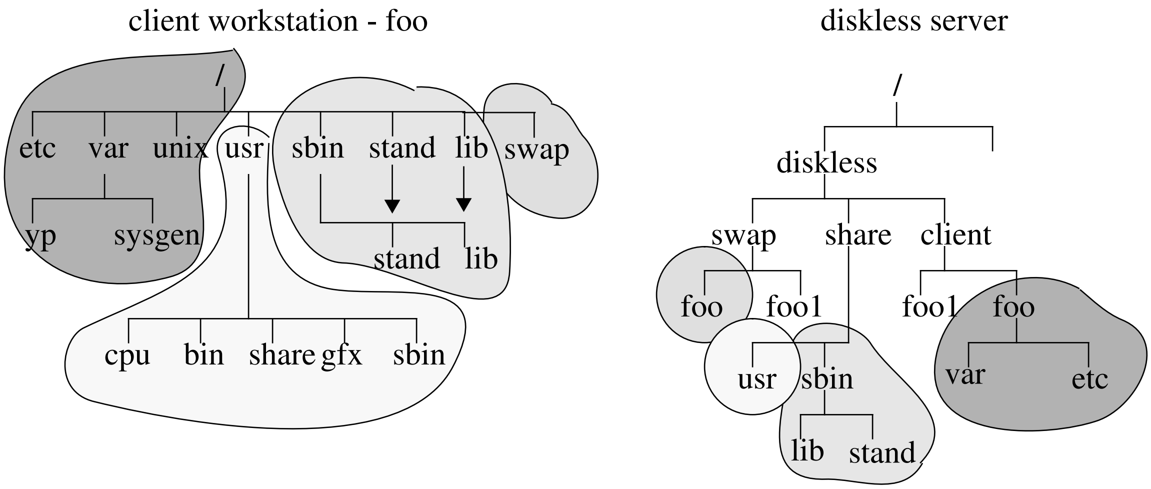
"`-S $SHARE`" tells *inst* that this is a client tree install and to install only those files not installed in the share tree rooted at `$SHARE`. "`-m$MACH`" are mach tags for this client. Other tags include `GFXBOARD`, `CPUARCH`, `CPUBOARD`, & `VIDEO`.

Following installation of idb files, *client_inst* cobbles up a local /etc/hosts file; chkconfigs nfs, network, verbose, & timed; lboots a unix kernel for the client; creates a /var/yp/ypdomain file; makes necessary symlinks like /lib -> sbin/lib and /stand -> sbin/stand; and creates an /etc/sys_id file.

3.3 Swap Tree

This has not changed from the past implementation. A swap file for each machine is placed in this tree. The size of the swap file determines the size of the swap partition on the client.

3.4 Mapping of client directories to diskless server directories



The above figure shows the mapping of client directories to directories on the server machine. Explicit mounts are:

```
/      ->     server:/diskless/client/foo
/sbin  ->     server:/diskless/share/sbin
/swap  ->     server:/diskless/swap/foo
/usr   ->     server:/diskless/share/usr
```

As stated previously, the first three mounts are performed when the kernel first starts running. /usr is mounted when /etc/bcheckrc is run. Note that the client tree shows symlinks from

```
/stand -> sbin/stand
/lib   -> sbin/lib
```

From the picture above, it should be clear that the /var and /etc directories reside on the root partition of the client. They are not separate mount points, and /var does not point to usr/var as on a diskfull machine.

3.5 Important facts to remember

Configuration files belong in /etc or /var. In general, configuration files have been moved to /etc with large files or directories being placed in /var. On a diskless workstation both directories are on the root partition. Remember that in a diskless environment, we are not concerned about the size of the root partition as much as maximizing sharing of non-root objects. On diskfull machines, however, the space allocated to root is important. This is why /var is a symlink to usr/-var on diskfull machines. By placing large configuration files or directories in /var, we move them off the root partition.

Given the split of files among two different trees, pre-ops, post-ops, and exit-ops must operate only on those files in that particular tree. For example, coding a post-op for the file /usr/lib/foo which needs to look at /etc/passwd will not work. Even if the file /etc/passwd were made available to the share tree, it would be only one /etc/passwd. Remember, there will be multiple /etc/passwd files in each client tree for one /usr/lib/foo file in the share tree. Running a post-op just once will not buy much.

inst now has an environment variable called **diskless** which has three values, depending on the type of installation. pre/post/exit-ops can query this variable and perform installation specific actions.

```
diskless=share|client|none
```

There is currently no easy-to-use mechanism to install patches or software which come in formats other than the *inst* format. Until an automated mechanism exists, packages can be unpacked on a diskless client which has mounted /usr read/write after the diskless server exports /usr read/write.

In general, please do not change pre/post/exit-ops to be no-ops when diskless installs are happening. If you have a problem, consult with someone more familiar with the diskless architecture.

4.0 Other information

This file is available in the RCS tree as ".../irix/cmd/diskless/diskless-arch.ps".

4.1 Revision history

Originally written by Michael Nishimoto (December, 1993)

4.2 Other documents

Diskless Workstation Administration Guide (document number: 007-0855-030)

"Directory Structure Re-organization in IRIX 5.x" available in the RCS tree as ".../irix/cmd/diskless/dir-reorg.fm"