GRUB AND THE SOLARIS™ OPERATING SYSTEM ON X86 PLATFORMS - A GUIDE TO CREATING A CUSTOMIZED BOOT DVD

John Cecere, Sun Services

Sun BluePrints™ OnLine — August 2006

SUN BLUEPRINTS [illegible] APPROVED

819-7546-10
Revision 1.0, 8/21/06
Edition: August 2006

TABLE OF CONTENTS

GRUB and the Solaris™ Operating System on x86 Platforms - A Guide to Creating a Customized Boot DVD

This Sun BluePrints™ OnLine article describes the open source GRand Unified Bootloader (GRUB) and its implementation for the Solaris™ Operating System on x86 Platforms. It provides a sequence of procedures that can be followed to customize a DVD using this framework.

This article contains the following sections:

Introduction

This article is a follow-on article to some of the material presented in the Sun BluePrints article titled *Creating a Customized Boot CD/DVD for the Solaris™ Operating System for x86 Platforms* (Cecere, John, and Fagerstrom, Dana, December 2005), which is available at the following URL: http://www.sun.com/blueprints/1205/819-3731.pdf

While there is some overlap of information between these two articles, both are complete and can be read independently of each other. This present article also contains additional information that pertains to a Solaris Jumpstart™ implementation.

When reading this article, be aware of the following considerations:

- Throughout the article, the term *DVD* is used. However, most of the information also applies to CDs, assuming that the desired result will fit on one.
- Unless otherwise indicated, the information in this article pertains to the Solaris OS for x86 Platforms only. No changes have been made to the boot process in the Solaris 10 1/06 OS on SPARC(r) platforms, and none of the procedures documented in this article apply to the Solaris OS for SPARC platforms.
- The procedures described in this article need to be performed on x86 systems running the Solaris 10 3/05 OS, or higher. The Solaris OS for x86 Platforms is required because the DVD miniroot that is to be manipulated is a little-endian Unix File System (UFS) and cannot be directly manipulated on a SPARC system (which is big-endian). The Solaris 10 03/05 OS or higher is required because the commands for manipulating objects within the Service Management Facility (SMF) do not exist in earlier Solaris OS releases.
- This article describes setting up a custom Jumpstart configuration on a DVD. However, it assumes that the user is already familiar with deploying the Solaris OS via Jumpstart (see http://docs.sun.com). It is not intended to be a tutorial on Jumpstart itself.

DCA Versus GRUB

This section reviews the Device Configuration Assistant (DCA) boot process used by previous versions of the Solaris OS for x86 Platforms and describes changes to the boot process in the Solaris 10 1/06 OS.

DCA

With the DCA, booting occurs in the following sequence of events:

1. BIOS loads the first sector of device 0x80 (*Master Boot Record*, or *MBR*) and executes. This loads `mboot`.
2. `mboot` finds the active partition, loads the first sector of the active partition, and executes. This loads `pboot`.
3. `pboot` locates, loads, and executes the boot block (`bootblk`), which is where the DCA is located.
4. DCA runs a real-mode environment to build the device tree and establishes communications with devices needed to boot via real-mode drivers. The real-mode environment is essentially DR DOS.
5. DCA loads and runs the Solaris kernel, passing off the device tree to it.
6. The Solaris kernel boots.

The Solaris 10 1/06 OS and GRUB

In previous versions of Solaris, there was a requirement to have a real mode to emulate some of the functions performed by Open Boot PROM in the SPARC architecture with the Device Configuration Assistant (DCA). This meant longer boot times and the need to write two drivers for every device involved in booting a system.

As of the Solaris 10 1/06 OS, the DCA and real-mode have been eliminated. The new boot process uses GNU GRUB to perform the bootstrap procedure. The kernel device tree is now built using the Advanced Configuration and Power Interface (ACPI) specification (see http://www.acpi.info/) that is implemented in the BIOS of virtually all new x86 motherboards today.

GRUB has been used for years in Linux and various versions of BSD as the standard file system-aware[1] boot loader for open source operating systems. GRUB's implementation in the Solaris OS is similar to the implementation in these other operating systems. One major difference in the Solaris OS implementation of GRUB is the ability to traverse a UFS, the standard file system used in the Solaris OS. The UFS code for GRUB was written by Sun and is available as open source via the Opensolaris™ initiative (see http://www.opensolaris.org/os/community/ufs/).

Disk Layout (Disk)

Because the Solaris OS for x86 Platforms—as of the Solaris 10 1/06 OS—now uses GRUB instead of the DCA, the layout of a hard disk (the boot components as they are laid out on a hard disk) has changed. In the following example, the Solaris OS is installed in fdisk partition 1 with a single (root) file system and a swap partition on disk `c0d0`. This example assumes that partition 1 is the only fdisk partition and that it uses all of the available space on the disk.

1.The term *file system-aware* here refers to the boot loader's ability to find data based on a file system structure rather than just a simple map of disk blocks.

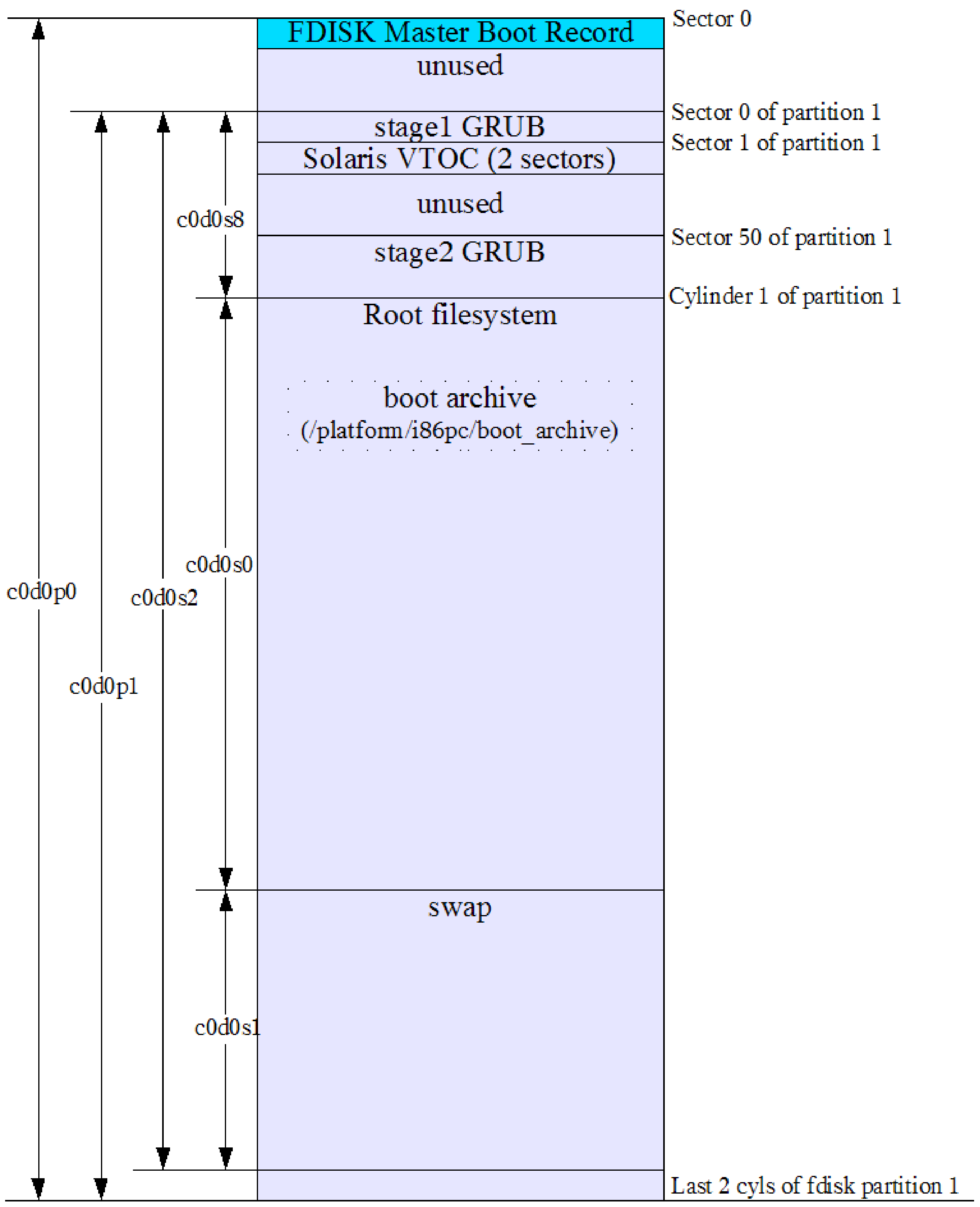

Figure 1. Disk Layout

As with all previous versions of the Solaris OS for x86 Platforms, the Solaris VTOC is embedded into the fdisk partition structure. Because of this, the Solaris OS for x86 Platforms looks at the fdisk partition that contains the partition ID of `0xbf` (or the previous `0x82`) as the whole disk.

The boot process occurs in the following sequence:

1. The BIOS loads the MBR.

   For the GRUB-based Solaris OS, this is the same as in DCA-based systems running the Solaris OS. The program loaded is normally `mboot`, but it can be stage1 GRUB instead. The `mboot` program merely finds the active partition, then loads and executes the first sector from it—the stage1 GRUB. The stage1 GRUB has the physical location of stage2 GRUB hard-coded within itself at offset `0x44` from the start of the partition boot sector. This address and the stage1 program itself are put there by the `installgrub` program.
2. Knowing the block address at which the stage2 GRUB starts on the disk, stage1 then loads and executes stage2. The stage2 GRUB contains code that allows it to navigate the UFS structure on the root file system.
3. The stage2 GRUB locates the GRUB configuration file (`/boot/grub/menu.lst`).
4. From here, a menu is presented to the user based on the contents of `/boot/grub/menu.lst`.

   Typically, for the Solaris OS, the `/boot/grub/menu.lst` file contains a kernel line and a module line. The module will point to the *boot archive*. The boot archive is a basic root file system that contains the kernel, the kernel modules, and a number of configuration files needed for bringing up the kernel (such as `/etc/system`).
5. The boot archive is loaded entirely in memory as a ramdisk.
6. The kernel is loaded from the boot archive.
7. The kernel then executes, finds out what the “real” root file system is (by looking at the `bootpath` property in the `/boot/solaris/bootenv.rc` file in the boot archive), mounts the device specified as `root`, and continues with the boot process.
8. The memory that was taken up by GRUB and the ramdisk are subsequently freed for future use.

Laying out all the components to make the boot process (described above) work properly is accomplished by running a program called `installgrub`. The `installgrub` program is a direct replacement for `installboot` and performs similar functionality. If run with the correct options, `installgrub` will perform the following operations:

1. Copy the stage1 GRUB to the partition boot sector (the first sector of the Solaris fdisk partition). The `installgrub` program modifies this copy of stage1 to point to the physical location of stage2 GRUB on the disk.
2. Copy the stage2 GRUB starting at an offset of 50 sectors from the beginning of the Solaris fdisk partition.
3. If the `-m` option is specified, copy the stage1 GRUB to the MBR.

As mentioned previously, the initial boot loader in the MBR is normally `mboot`, which is the same `mboot` program that was used for booting DCA-based Solaris systems. The `mboot` program is put there by the fdisk program whenever it rewrites the MBR to update the partition table. It is there by default because the Solaris installation program runs fdisk to partition the disk and uses `installgrub` without the `-m` option.

It is possible to use stage1 GRUB in its place by running `installgrub -m`. However, if fdisk is subsequently run and a partition change is made, it will be overwritten with `mboot`.

Having described where the components go, the next sections describe what the components are for.

fdisk Master Boot Record (MBR)

The first 512 bytes of any disk in the x86 architecture is what is commonly called the Master Boot Record (MBR). This sector contains the primary boot loader for booting the system. In the Solaris OS for x86 Platforms, this can be either `mboot` or stage1 GRUB, but is `mboot` by default. Either way, the function of the primary boot loader is to locate the active partition, then load and execute a program contained in sector 0 of that partition (which, in the Solaris OS, will be stage1 GRUB).

The MBR also contains the fdisk partition table, which contains four 16-byte entries, one for each possible primary partition.[2] Out of these 16 bytes, the fields that impact the boot process are:

- active flag
- system type
- relative sector
- number of sectors

The active flag field denotes whether this partition is the boot partition. It can contain the value `0` (not a boot partition) or `0x80` (boot partition). The primary boot loader will look for the first partition on the disk with this flag set to `0x80` and attempt to boot from it.

The system type field contains a one-byte tag that identifies what the partition is being used for. No standards body was ever established to define values for this field. As a result, several cases exist in which a specific value has more than one meaning. The Solaris OS for x86 Platforms is unfortunately affected by this. Prior to the Solaris 10 release, Solaris partitions had a type of `0x82`, which also happens to be the type of a Linux swap partition. This causes problems when installing both operating systems on a single disk, as each OS wants to use the partition tagged with `0x82` for its own purpose. With the release of Solaris 10, Sun has officially changed the partition type code to `0xbf` (decimal 191). For more information about the values used for x86 partition types, refer to the following URL:

http://www.win.tue.nl/~aeb/partitions/partition_types-1.html

2.The Solaris OS for x86 Platforms does not currently support extended partitions.

stage 1 GRUB

The stage1 GRUB can be either a primary boot loader, a partition boot program, or both. If the stage1 GRUB is a primary boot loader, then it merely loads and executes the first sector of the active partition as `mboot` does. In the Solaris OS for x86 Platforms, this partition boot program would also be stage1 GRUB. As a partition boot program, stage1 GRUB is responsible for loading stage2 GRUB from a hard-coded offset within the partition that stage1 GRUB occupies.

stage 2 GRUB

The stage2 GRUB contains code that allows it to navigate a file system. For the Solaris OS, this allows it to find components of the boot process within the root file system. Normally, the two components that it needs to locate are:

- the configuration file (`/boot/grub/menu.lst`)
- the boot archive (`/platform/i86pc/boot_archive`, or miniroot for a DVD)

Although the mechanism for retrieving the kernel differs slightly, the stage2 GRUB program in the Solaris OS for x86 Platforms is analogous to the `ufsboot` program in the Solaris OS for SPARC platforms. The `ufsboot` program retrieves the kernel directly from the file system, whereas the stage2 GRUB retrieves the module (boot archive or miniroot) from the file system first, then retrieves the kernel from the module after the module has been unbundled and loaded into memory.

Solaris VTOC

In the Solaris OS for SPARC platforms, the layout of a hard disk is defined solely by the Solaris Volume Table of Contents (VTOC). In contrast, the Solaris OS for x86 Platforms uses the fdisk partitioning scheme with the VTOC embedded within an fdisk partition. The VTOC on a SPARC system is on the first block of the disk, and this block contains no boot code. The boot code on SPARC systems (referred to as the boot block) is contained in blocks 1-15.

The structure of the VTOC itself in the Solaris OS for x86 Platforms differs somewhat from its SPARC counterpart. In the Solaris OS for x86 Platforms, the VTOC is able to accommodate 16 partitions, not just 8. This might not be immediately evident, as the `format` command does not currently provide the ability to manipulate partitions 8-15, although `prtvtoc` and `fmthard` do.

Other differences in the Solaris OS for x86 Platforms include:

- There are two sectors allocated for the VTOC instead of just one.
- The VTOC is physically located directly after the partition boot sector of the fdisk partition on which it resides, rather than at the beginning of the disk.

The VTOC contains information about the size, location, and type of slices on the disk, along with information about the disk geometry. In the case of the Solaris OS for x86 Platforms, the disk geometry really just refers to the geometry of the fdisk partition on which it resides. In effect, this makes the Solaris OS for x86 Platforms fdisk partition look like the entire disk to the VTOC.

The following table shows a summary of the components required for booting the Solaris OS for x86 Platforms using GRUB.

Table 1. Boot Components Layout

Component	**Start**	**Length**	**Contents**
fdisk Master Boot Record	Sector `0` of the disk	One sector	**Bytes 0-445:** Boot code (`/usr/lib/fs/ufs/mboot`) **Bytes 446-509:** partition table **Bytes 510-511:** Signature `0xaa55`
stage1 GRUB	Sector `0` of the fdisk partition	One sector	`/boot/grub/stage1`
Solaris VTOC	Sector `1` of the fdisk partition	Two sectors	Disk label, Volume Table of Contents
stage2 GRUB	Sector `50` of the fdisk partition	Can occupy up to the end of the first cylinder of the fdisk partition	`/boot/grub/stage2`
VTOC partition space	Cylinder `1` of the fdisk partition	Up to two cylinders before the end of the fdisk partition	File systems, swap, raw devices, and so on

Solaris Disk Device Paths

In the Solaris OS for SPARC platforms, where VTOC is the only partitioning scheme used, the partitions (or slices) are referred to by the `s` value in the device path. For example, `/dev/dsk/c0t0d0s0` refers to the first partition on the disk. This does not change for the Solaris OS for x86 Platforms, but because there is a second partitioning scheme in which the VTOC is embedded, some differences need to be clarified. Normally, in the Solaris OS for SPARC platforms, slice 2 refers to the entire disk. In the Solaris OS for x86 Platforms, slice 2 refers to everything on the disk that can be addressed using the Solaris VTOC structure. Specifically, this is the entire fdisk partition on which the Solaris OS for x86 Platforms resides, minus the last two cylinders.[3]

3.These two cylinders were used in the past for alternate cylinders but are no longer used as such in the Solaris 10 OS. They are still there for backward compatibility.

The Solaris OS for x86 Platforms also has device path names to handle access to the fdisk structure of a disk, as shown in the following table.

Table 2. fdisk Solaris Device Path Names

Component	**Solaris Device Paths (block device)**	**Solaris Device Paths (character device)**
Entire Physical Disk	`/dev/dsk/c0t0d0p0`	`/dev/rdsk/c0t0d0p0`
fdisk partition 1	`/dev/dsk/c0t0d0p1`	`/dev/rdsk/c0t0d0p1`
fdisk partition 2	`/dev/dsk/c0t0d0p2`	`/dev/rdsk/c0t0d0p2`
fdisk partition 3	`/dev/dsk/c0t0d0p3`	`/dev/rdsk/c0t0d0p3`
fdisk partition 4	`/dev/dsk/c0t0d0p4`	`/dev/rdsk/c0t0d0p4`

The device names assigned to SCSI disks are the same as those in the Solaris OS for SPARC platforms in that they both use the `c?t?d?s?` (controller,target,device,slice) convention. However, x86 systems typically come with ATA controllers and disks. ATA does not maintain the concept of targets, and a controller is capable of addressing only up to four devices. The following table shows the mapping of devices on an ATA controller to the Solaris OS for x86 Platforms device paths.[4]

Table 3. ATA Device Path Names

Function	**Solaris Device Paths (block device)**	**Solaris Device Paths (character device)**
Primary Master	`/dev/dsk/c0d0p0`	`/dev/rdsk/c0d0p0`
Primary Slave	`/dev/dsk/c0d1p0`	`/dev/rdsk/c0d1p0`
Secondary Master	`/dev/dsk/c1d0p0`	`/dev/rdsk/c1d0p0`
Secondary Slave	`/dev/dsk/c1d1p0`	`/dev/rdsk/c1d1p0`

Disk Layout (DVD)

Because the Solaris OS for x86 Platforms now uses GRUB instead of the DCA, the layout of a DVD (the boot components as they are laid out on a DVD) has changed. The following figure shows the basic layout of the GRUB-based Solaris OS for x86 Platforms boot DVD:

4.The exception to this is that ATA CDROM/DVD devices in both SPARC and x86 use the sd driver and therefore have targets associated with them.

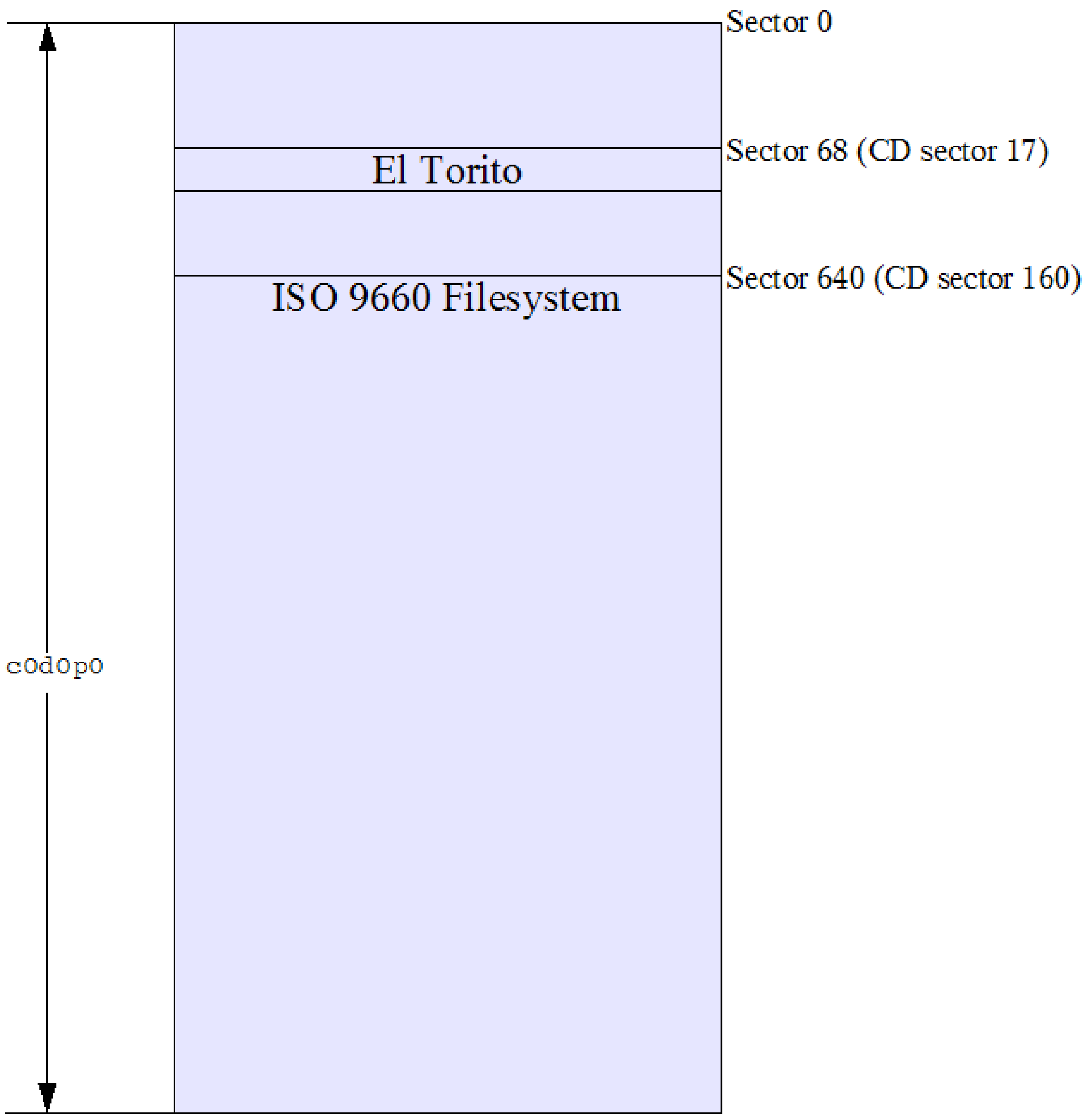

Figure 2. DVD Layout

The layout of the GRUB-based Solaris OS boot DVD is much simpler than that of previous DCA-based versions of the Solaris OS. The structure of the DVD is no different from a normal (bootable) data DVD with a single ISO file system on it, and it does not require either the fdisk or VTOC partitioning schemes. Because of this, it is now possible to set up the Solaris OS for x86 Platforms boot server on a SPARC system directly from the DVD. This was not possible with DCA-based versions of the Solaris OS for x86 Platforms due to endian incompatibilities between SPARC and x86 systems. Still, the procedure to customize the DVD (see “Tasks for Customizing a GRUB-based Solaris OS for x86 Platforms DVD” on page 12) requires it to be done on a Solaris OS for x86 Platforms system because the miniroot on the DVD uses little endian byte ordering and cannot be directly manipulated on a big endian SPARC system.

El Torito

The boot process for the Solaris OS for x86 Platforms DVD is handled by El Torito, the industry standard boot mechanism for DVDs. The El Torito specification for CDROM booting was designed by Phoenix Technologies as a way to implement a bootable CDROM without needing to make drastic changes to the BIOS. The El Torito information starts at CD sector 17 (2048*17 = byte 34816), which comes before cylinder 1 on the CD where the ISO 9660 file system is located.

El Torito defines, among other things:

- a boot catalog to allow multiple choices from which to boot
- a boot record volume descriptor (`brvd`), which is a special volume descriptor on the CD to identify the beginning of the boot catalog
- a validation entry as a means to validate that what is actually being looked at is a boot catalog
- an initial/default entry to give information on what needs to be loaded

The boot program used by GRUB for El Torito booting is `/boot/grub/stage2_eltorito`. The El Torito specification also applies to data DVDs using ISO9660 file systems.

For more information on El Torito, go to the following URL:

http://www.phoenix.com/NR/rdonlyres/98D3219C-9CC9-4DF5-B496-A286D893E36A/0/specscdrom.pdf

Miniroot

The miniroot on a GRUB-based Solaris OS for x86 Platforms DVD is similar to the boot archive on a Solaris OS hard disk. The miniroot contains the kernel, modules, and configuration files needed to boot. However, the miniroot on the DVD is more comprehensive than the `boot_archive` and actually contains a functional, stand-alone root file system that boots to the Solaris OS installation program. When the Solaris OS boots from DVD, it will never find the “real” root file system to mount because the `bootpath` argument in `/boot/solaris/bootenv.rc` is left unset. This tells the kernel to use whatever it already has for the root file system, so it will use the miniroot itself.

Both the miniroot and a disk-based `boot_archive` are similar in that they are both a compressed image of a file system. However, the file system types differ. By default, in the Solaris 10 1/06 OS release, the boot archive is an HSFS, whereas the miniroot is a UFS. In either case, the GRUB stage2 is responsible for uncompressing them and laying them out in memory as a file system.

Tasks for Customizing a GRUB-based Solaris OS for x86 Platforms DVD

There are two parts to customizing a GRUB-based Solaris OS for x86 Platforms DVD.

- **Customizing the miniroot.** The miniroot will be the root file system when the DVD is booted. The main difference between the miniroot used by GRUB and the miniroot used by the DCA is that the GRUB miniroot is a file system image that gets laid out entirely in memory. The miniroot actually exists as a file on the DVD, and GRUB handles the process of extracting it to memory as a ramdisk. In contrast, the DCA miniroot is an actual file system that occupies a separate VTOC slice on the DVD.
- **Customizing the DVD**, which pertains to everything else on the DVD. The most obvious modification to make to the DVD is to put a flash archive on it for doing Solaris OS installations.

The basic steps for putting together a customized, GRUB-based Solaris OS for x86 Platforms boot DVD are:

- Step 1: Copy the DVD to Writable Disk
- Step 2: Modify or Rebuild the miniroot
- Step 3: Add Customizations to the On-disk Copy of the DVD
- Step 4: Recreate the ISO Image

The examples in this section use specific names for the files and directories to be created. You can substitute different names as long as they are consistently used throughout the applicable procedure.

Table 4. File Names and Folders Used in These Examples

Folder/File	Description
`/cdrom/cdrom0` or `/mnt`	Initial mount point for the DVD or DVD ISO image.
`/var/tmp/dvd`	Where to copy the DVD directory structure to (the on-disk copy of the DVD). This is ultimately where you will create the customized ISO image from.
`/var/tmp/mrfile`	Name of the miniroot file that is created from the DVD miniroot.
`/tmp/mrmnt`	Where to mount the miniroot file system image.
`/var/tmp/mrdir`	Where to copy the miniroot directory structure to (if necessary). This will be used as a staging area for the customized miniroot.

Note – The following steps should be performed as the root user on an x86 system running the Solaris 10 OS or higher.

Step 1: Copy the DVD to Writable Disk

The process of copying the DVD to writable disk is relatively straightforward. You need enough space on the target disk to store the copy (~3GB for the Solaris 10 1/06 OS release). You just use `cpio` or `tar` to perform the copy. Begin by mounting the DVD, either using `vold` or by manually mounting it.

```
# mount -F hsfs -o ro /dev/dsk/c1t0d0p0 /mnt
```

The above example assumes that your DVD drive is `/dev/dsk/c1t0d0p0`. This step is obviously not necessary if you use `vold` to mount the DVD.

Alternatively, you can use an ISO image file of the DVD (such as a `dd` copy of the DVD) and mount it using `lofiadm`, as shown in the following example (which assumes that the ISO image file is in the current directory).

```
# lofiadm -a `pwd`/sol-10-u1-ga-x86-dvd.iso
/dev/lofi/1
# mount -F hsfs -o ro /dev/lofi/1 /mnt
```

Next, create a directory on the target file system, and then perform the copy operation. Make sure that you have enough space on the target file system to which you are copying.

```
# mkdir /var/tmp/dvd
# cd /mnt
```

Note – If you are using `vold`, then run `cd /cdrom/cdrom0` instead of `cd /mnt`.

```
# find . -depth -print | cpio -pdm /var/tmp/dvd
6539088 blocks
# cd /; umount /mnt
# lofiadm -d /dev/lofi/1 (if you used an ISO image file rather than the actual DVD)
```

Note – This operation might take some time to copy several gigabytes of data.

Step 2: Modify or Rebuild the miniroot

The miniroot refers to the root file system image on the DVD. DCA-based versions of the Solaris OS for x86 Platforms had the miniroot as a separate slice on the CD and mounted that slice as the root file system at boot time. With the GRUB-based Solaris OS for x86 Platforms, the miniroot exists on the DVD as the `boot/x86.miniroot` file. This file is actually a compressed (with gzip) file system image. It gets extracted into memory by GRUB and mounted as a ramdisk device at boot time.

Without delving extensively into the details on the structure of the miniroot, you should understand key aspects of it. The miniroot initially gets created in much the same way that the Solaris OS gets installed on a system, with the following two main differences:

- A miniroot is structured around the fact that the root file system will be read-only. Because of this, certain things that normally exist as regular files and directories in the Solaris OS exist as symbolic links into `/tmp` so that they can be written to after booting.
- Because the basic purpose of the miniroot is fairly limited, the miniroot itself is limited. Certain functionality (such as 64-bit support) has been intentionally omitted or removed from the standard miniroot after creation. This is to allow the ability to boot on systems that have limited memory resources (recall that the miniroot gets loaded entirely into memory). Although 64-bit support was removed, this does not mean that the miniroot cannot be used to install a 64-bit system. Instead, it means that the

miniroot itself cannot be booted in 64-bit mode. It is possible to create a miniroot with 64-bit support; however, describing the steps to do so are outside the scope of this article.

In this article, the process of manipulating the miniroot is divided into a number of procedures:

- Start—Extract the miniroot
- Procedure 1—Stage the miniroot
- Procedure 2—Customize the miniroot
- Procedure 3—Handle Large miniroots
- Procedure 4—Destage the miniroot
- Finish—Copy the miniroot Back

The procedures you follow depend on what you need to do to the miniroot.

- If modifications will not add more than (approximately) 1MB to the miniroot, then the process is a simple matter of extracting the miniroot, modifying it, and copying the miniroot back.

  However, because the file system (image) on which the miniroot resides has just enough space to hold the original contents, the file system might need to be created from scratch if more data needs to be added to it.

- If more than (approximately) 100MB needs to be added, then it might be necessary to offload `/usr` (or any other directories) to the main area of the DVD and have it mounted as a separate file system at boot time. In this last case, you need to make some extra modifications to the miniroot so that it is able to find the file system and mount it.

The flow chart in the following figure shows the general logic used to determine which of these procedures to use. The Start and Finish procedures will always be performed because these procedures describe how to initially extract and subsequently rebundle the miniroot.

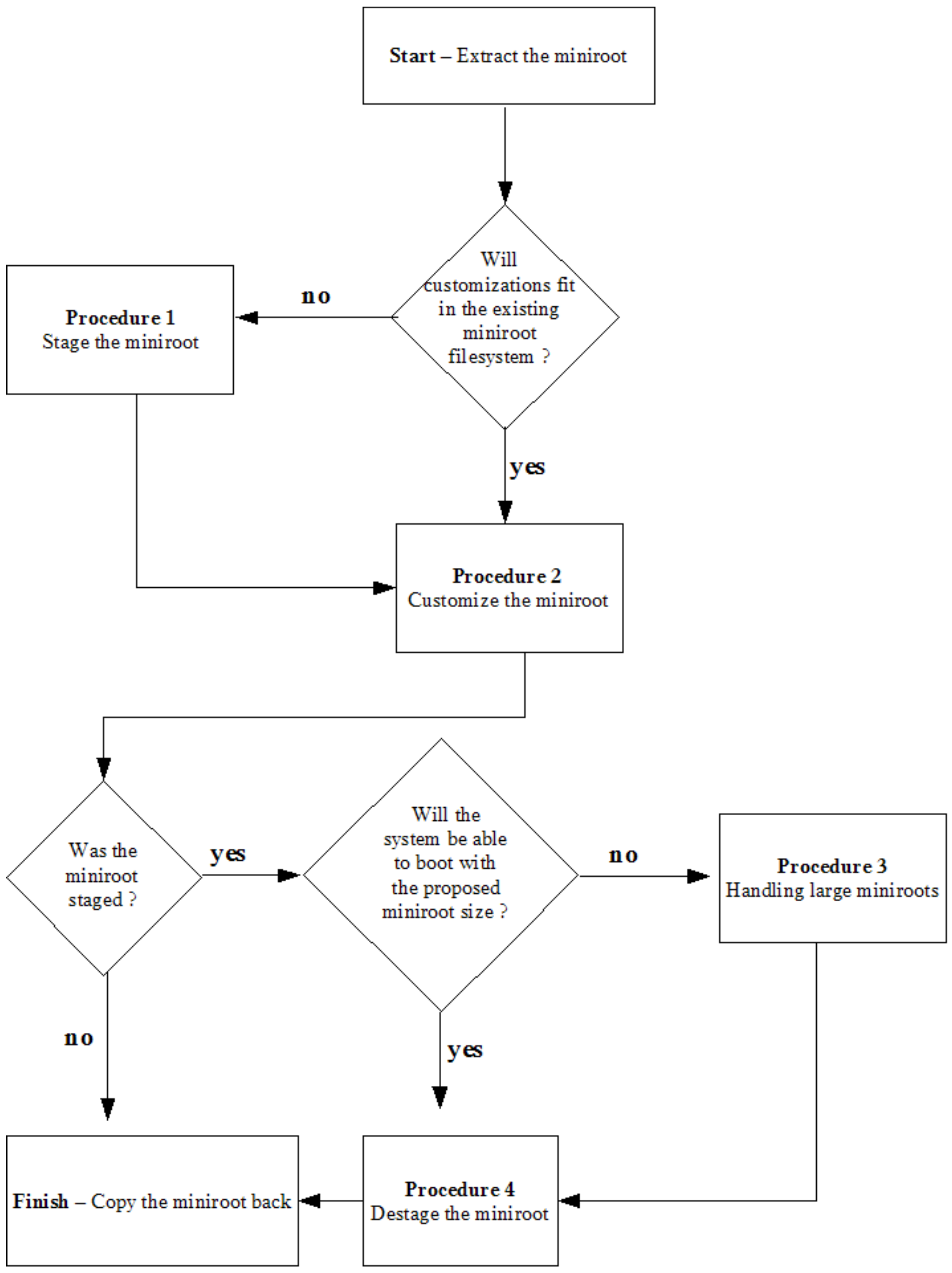
Start – Extract the miniroot
Will customizations fit in the existing miniroot filesystem ?
no
Procedure 1 Stage the miniroot
yes
Procedure 2 Customize the miniroot
Was the miniroot staged ?
yes
Will the system be able to boot with the proposed miniroot size ?
no
Procedure 3 Handling large miniroots
no
yes
Finish – Copy the miniroot back
Procedure 4 Destage the miniroot

Figure 3. Miniroot Customization Flow

To determine whether a system will boot with any given miniroot size, perform the following general calculation:

total memory needed to boot	=	uncompressed miniroot size
	+	kernel memory
	+	sum of space used in all tmpfs file systems
	+	memory used by system processes

For example, the following observations were made while booting a Sun Fire™ V20z Server with 4GB of memory from the Solaris 10 1/06 OS DVD:

Uncompressed miniroot size	130MB
Kernel memory	40MB
Sum of space used in all tmpfs file systems	2MB
Memory used by system processes	22MB
Total[a]	194MB

a. This total does not take into account the memory and tmp space needed to run anything else (such as the Solaris OS installation program).

As shown in the table above, a system with only 256MB of memory would be unable to boot if the miniroot size were increased by 62MB (uncompressed). In this and similar cases, consider the instructions in "Procedure 3—Handle Large miniroots" on page 19.

Start—Extract the miniroot

Before performing any work on the miniroot, extract and mount it using the following commands:

```
# gzcat /var/tmp/dvd/boot/x86.miniroot > /var/tmp/mrfile
# lofiadm -a /var/tmp/mrfile
/dev/lofi/1
# mkdir /tmp/mrmnt
# mount /dev/lofi/1 /tmp/mrmnt
```

Procedure 1—Stage the miniroot

If you need to add packages or do anything else that will cause the miniroot's file system to run out of space, then you need to create a new file system with a sufficient amount of space, and then copy the existing file system contents to it. Before you can do this, you need to copy the directory structure from the miniroot to some other location. The following example uses the `/var/tmp/mrdir` directory as this staging area.

```
# mkdir /var/tmp/mrdir
# cd /tmp/mrmnt; find . -depth -print | cpio -pdm /var/tmp/mrdir
294048 blocks
# cd /;umount /tmp/mrmnt; lofiadm -d /dev/lofi/1; rm /var/tmp/mrfile
# cd /var/tmp/mrdir; bzcat /var/tmp/dvd/Solaris_10/Tools/Boot/pkg_db.cpio.bz2 | cpio -icdm
```

```
14410 blocks
```

The final command restores the package database so that you can add patches and packages to the miniroot.

Once this procedure is complete, you can make modifications to the miniroot in the staging area.

Procedure 2—Customize the miniroot

If the miniroot was staged (as described in “Procedure 1—Stage the miniroot” on page 16), then you will perform customizations in the staging area (`/var/tmp/mrdir`). Otherwise, the customizations can be performed directly on the mounted miniroot image (`/tmp/mrmnt`).

This section describes two examples of miniroot customizations:

- “Example 1—Setting Up for Jumpstart” on page 17 shows setting up a completely self-contained DVD to perform a Jumpstart installation
- “Example 2—Adding a Package or Patch to the miniroot” on page 18 shows how to add a package and patch.

Example 1—Setting Up for Jumpstart

Modifying the miniroot for Jumpstart normally does not require staging it, as you need to make only two modifications:

1. Remove the existing symbolic link from the miniroot, and then place the `sysidcfg` file in `/etc`.

   ```
   # rm /tmp/mrmnt/etc/sysidcfg
   # cp sysidcfg /tmp/mrmnt/etc
   ```

2. Modify the boot script (`/tmp/mrmnt/usr/sbin/install.d/profind`) so that it will pick up a customized Jumpstart configuration. This involves removing the conditional for the existence of `/tmp/.preinstall`:

   ```
   cdrom()
   {
       # Factory JumpStart is only allowed with factory
       # stub images, indicated by the file /tmp/.preinstall
       #
       if [ -f /tmp/.preinstall ]; then <---------------------------------- Remove this line
           mount -o ro -F lofs ${CD_CONFIG_DIR} ${SI_CONFIG_DIR} >/dev/null 2>&1
   T
           if [ $? -eq 0 ]; then
               verify_config "CDROM"
           fi
       fi <----------------------------------------------------------------- Remove this line
   }
   ```

These are normally all the modifications that you need to make to the miniroot in order to set up a custom Jumpstart. However, if you want to perform a flash install, then you need to be aware of the following bug:

```
Bug ID: 6259653
Synopsis: cp command fail to copy 3.2 GB flash archive from DVD in S9/S10 32 bit x86 system.
```

Note – Check the current version of your Solaris OS to determine whether this bug is still open.

This bug prevents the Solaris OS from being able to read a file greater than 2GB from a HSFS (ISO 9660 file system) with a 32-bit kernel. If the flash archive is larger than 2GB, try using compression (the `-c` option to `flarcreate`), if you have not already done so.

If, after compressing the flash archive, the size is still greater than 2GB, then you can use a workaround until a patch is released for the Solaris 10 OS. Because this issue was fixed in build 33 of the OpenSolaris operating system, the workaround is to replace the HSFS driver in your miniroot (`/kernel/fs/hsfs`) with the HSFS driver in your current version of Opensolaris:

1. Download the first CD image from the pre-built archives of Solaris Express.

   Opensolaris downloads are available from http://www.opensolaris.org/os/downloads/on/.
2. Unzip the image file.
3. Use `lofiadm` and `mount` to gain access to the miniroot, as described earlier for the Solaris 10 OS DVD.
4. Extract the Opensolaris miniroot using the instructions provided in “Start—Extract the miniroot” on page 16.
5. Copy `/kernel/fs/hsfs` from the Opensolaris miniroot to the comparable location on your customized miniroot.
6. Because the miniroot modifications for Jumpstart do not require staging the miniroot, you can proceed from here to “Finish—Copy the miniroot Back” on page 22.

Thereafter, modifications need to be made to the DVD image itself for a custom Jumpstart, as described in “Step 3: Add Customizations to the On-disk Copy of the DVD” on page 22.

Example 2—Adding a Package or Patch to the miniroot

To add a package to the miniroot, you will most likely need to increase the miniroot file system size. Therefore, you will need to follow the instructions in “Procedure 1—Stage the miniroot” on page 16, and “Procedure 4—Destage the miniroot” on page 20. This includes restoring the package database. Once the miniroot is in the staging area (`/var/tmp/mrdir`) and the package database has been extracted correctly, you can proceed with the package or patch installation.

The syntax for `pkgadd` is:

```
# pkgadd -R /var/tmp/mrdir -d . <package name>
```

This syntax assumes that the package directory is in the current working directory.

To install a patch in the miniroot, the syntax for `patchadd` is:

```
# patchadd -C /var/tmp/mrdir <patch directory>
```

Note that, as of the time when this article was written, a bug affects the use of `patchadd -C`:

```
Bug ID: 6366310
Synopsis: 118844-28/118822-27 or 28 patchadd -C issue with SUNWcsr and /var
```

The issue with this bug is that, on a miniroot, `/var` exists as a symbolic link to `/tmp/root/var`. This structure needs to remain intact if anything under `/var` is to be writable when the miniroot is booted. The problem is that `patchadd` removes this symbolic link and replaces it with a `/var` directory when it encounters a patch that needs to patch something under `/var`.

The workaround for this bug is to do the following before running `patchadd`:

```
# PKG_NONABI_SYMLINKS="true"
# export PKG_NONABI_SYMLINKS
```

for sh or ksh, or

```
% setenv PKG_NONABI_SYMLINKS true
```

for csh.

After completing your package or patch installation, use the flowchart in Figure 3, "Miniroot Customization Flow" on page 15, to determine whether you need to complete Procedure 3—Handle Large miniroots. If not, proceed to "Procedure 4—Destage the miniroot" on page 20 and then implement "Finish—Copy the miniroot Back" on page 22.

Procedure 3—Handle Large miniroots

Should the miniroot become abnormally large, you might need to complete additional tasks before you finish customizing the miniroot. With the GRUB-based Solaris OS for x86 Platforms, the miniroot is an image file that gets unbundled and entirely loaded into memory. Because of this, you need to be aware of some restrictions.

- The system on which you are booting needs to have enough memory to hold the kernel and the entire uncompressed miniroot, plus memory for user consumption. The standard Solaris 10 1/06 OS miniroot is ~135MB uncompressed. When you consider that the minimum memory requirement to install and boot the Solaris 10 1/06 OS is 256MB, you can make a reasonable estimate about what the system will need based on the size of the miniroot and the formula described in Step 2.
- Although the other restriction is outside the scope and context of this article, it is worth mentioning. If you are building the DVD for a network boot server, there is a hard limitation on how much data (64K-1 packets) `tftp` will download for a PXE boot. The `pxegrub` program will use 1492 byte packets for the download of the miniroot, which means that the miniroot can be no larger than 1492*65535, or ~93MB compressed.

One way to get around the first restriction is to move the `/usr` directory out of the miniroot and into the `boot` directory on the DVD, leaving it as a directory structure on the DVD itself. This takes up more room on the DVD, but it cuts down the size of the miniroot substantially. To help you do this, the author has provided a downloadable compressed file (available from the Sun Download Center) containing a startup script, an SMF manifest entry, and a binary program. The startup script will run early in the boot process (as defined by the manifest entry) and execute the binary to locate and mount `/usr` from the DVD before the rest of the system comes up. To obtain the downloadable file, go to the following URL:

http://javashoplm.sun.com/ECom/docs/Welcome.jsp?StoreId=8&PartDetailId=GRUB-SOL-x86-G-F&TransactionId=try

This download contains the following files:

Table 5. Files in the Download

File	**Description**
`findbdev`	Binary to locate a CD/DVD drive node in `/devices` with a readable disk in it.
`mntusr`	Shell script that mounts `/usr` from the DVD using the `findbdev` program.
`mntusr.xml`	Manifest entry for `mntusr`. Inserts itself as a dependency of install-discovery so it will be the first thing to run after the `svc.startd` starts.

As mentioned previously, the strategy is to move the `/usr` directory to the boot directory on the DVD. When the DVD is booted, the boot process will first mount the cdrom device itself as `/cdrom`, then do a `lofs mount` from the directory containing `/usr` to the actual `/usr` directory.

Note – You cannot mount the `boot/usr` directory directly to `/usr` because `boot/usr` is not a block device.

After completing your customizations, perform the following procedure on the miniroot.

1. Download the zip file and extract it into the current working directory (which should not exist anywhere within `/var/tmp/mrdir` or `/var/tmp/dvd`).
2. Run the following commands:

```
# cp findbdev /var/tmp/mrdir/sbin
# chmod 755 /var/tmp/mrdir/sbin/findbdev
# cp mntusr /var/tmp/mrdir/lib/svc/method
# chmod 755 /var/tmp/mrdir/lib/svc/method/mntusr
```

3. Modify the SMF repository to run the script, then move the `/usr` directory.

```
# echo "repository /var/tmp/mrdir/etc/svc/repository.db" > /tmp/svc.in
# echo "import mntusr.xml" >> /tmp/svc.in
# svccfg -f /tmp/svc.in
# mv /var/tmp/mrdir/usr /var/tmp/dvd/boot
# mkdir /var/tmp/mrdir/usr
```

4. Copy the `lofs mount` command to the `/usr` directory.

   This is needed because the `/sbin/mount` command is capable of mounting file systems that are supported for booting only (such as UFS, NFS, and HSFS). Remember that nothing in `/usr` is available (such as `/usr/sbin/mount`, which can mount any type of file system supported in Solaris) at the time you need to perform this mount. To get around this limitation, run the following command.

```
# cp -p /var/tmp/dvd/boot/usr/lib/fs/lofs/mount /var/tmp/mrdir/sbin/mount_lofs
```

   The `mntusr` script will expect the `mount_lofs` binary to be in `/sbin`.

Procedure 4—Destage the miniroot

If you intend to use this miniroot to add more packages or patches in the future, then you might want to preserve the package database after you have finished and rewrite it back to the DVD directory so that it is updated on your customized DVD. The following commands will save the new package database to the original location.

```
# cd /var/tmp/mrdir
# find tmp/root/var/sadm/install tmp/root/var/sadm/pkg -print | cpio -ocdm >
     /tmp/pkg_db.cpio
14416 blocks
# cat /tmp/pkg_db.cpio | bzip2 - > /var/tmp/dvd/Solaris_10/Tools/Boot/pkg_db.cpio.bz2
```

For most cases, this will not be necessary because you can always go back to the original package database and just reinstall all of the extra packages.

Whether you preserve your updated package database or not, you will want to remove it from the miniroot because it takes up space and serves no purpose for booting the DVD. In fact, because of the way that a miniroot is loaded on a DVD boot, it will not even be accessible after booting the DVD because a `tmpfs` will be mounted over it. To remove the two directories that comprise the software database:

```
# rm -r /var/tmp/mrdir/tmp/root/var/sadm/install
# rm -r /var/tmp/mrdir/tmp/root/var/sadm/pkg
```

Now that the miniroot customizations are complete, you need to create a file system image large enough to hold it. Start by calculating how large the file system needs to be in order to hold the contents of the customized miniroot.

```
# cd /var/tmp/mrdir
# du -sk .
123799
```

In the preceding example, the miniroot directory takes up 123799k. Add to this number about 5-10MB to for file system overhead. Round it off to 130MB to simplify calculations.

```
# mkfile 130m /var/tmp/mrfile
# chmod 644 /var/tmp/mrfile
# lofiadm -a /var/tmp/mrfile
/dev/lofi/1
```

Now that you have a block device sufficiently large to hold the miniroot, create a file system on it and copy the directory contents into the device.

```
# newfs -i 8192 -m 1 /dev/rlofi/1
newfs: construct a new file system /dev/rlofi/1: (y/n)? y
/dev/rlofi/1:   265800 sectors in 443 cylinders of 1 tracks, 600 sectors
        129.8MB in 28 cyl groups (16 c/g, 4.69MB/g, 576 i/g)
super-block backups (for fsck -F ufs -o b=#) at:
 32, 9632, 19232, 28832, 38432, 48032, 57632, 67232, 76832, 86432,
 172832, 182432, 192032, 201632, 211232, 220832, 230432, 240032, 249632, 259232
# mount /dev/lofi/1 /tmp/mrmnt
# cd /var/tmp/mrdir; find . -depth -print | cpio -pdm /tmp/mrmnt
294048 blocks
```

In the preceding example, note the use of `/dev/rlofi`, which is the character device interface and is needed for `newfs`.

Finish—Copy the miniroot Back

Regardless of which procedure you followed in this article, you need to complete the following steps to put the miniroot back into the DVD image.

```
# umount /tmp/mrmnt
# lofiadm -d /dev/lofi/1
# gzip /var/tmp/mrfile
# mv /var/tmp/mrfile.gz /var/tmp/dvd/boot/x86.miniroot
```

Step 3: Add Customizations to the On-disk Copy of the DVD

After modifying the miniroot and putting it back on the DVD, it might be necessary to make modifications to the DVD image itself. The most obvious need to do this would be if you used the example in "Procedure 1—Stage the miniroot" on page 16 to create a custom Jumpstart DVD. The `sysidcfg` file has been installed, and you have modified the `profind` script to unconditionally mount `SI_CONFIG_DIR`, which will be mounted from the `.install_config` (`/var/tmp/dvd/.install_config`) directory in the top-level directory of the DVD. This is where the `rules.ok` file, profiles, begin scripts, and finish scripts need to go.

If you are doing a flash installation, then the flash archive can go anywhere within the DVD directory hierarchy, as long as the reference is prefaced by `/cdrom` and the `local_file` keyword is used. For example, if the flash archive (`myflash.flar`) is placed under `/var/tmp/dvd/Solaris_10`, then the appropriate line in the profile to specify the location would be:

```
archive_location local_file /cdrom/Solaris_10/myflash.flar
```

To allow for more space for the flash archive, the `Product` and `UpgradePatches` directories can safely be removed from `/var/tmp/dvd`.

```
# rm -r /var/tmp/dvd/Solaris_10/Product
# rm -r /var/tmp/dvd/Solaris_10/UpgradePatches
```

Note – Remove these two directories *only* if you are doing a flash installation. An upgrade or initial installation will require these directories to be present.

To make the customized DVD perform a hands-off Jumpstart, you need to make one final modification on the DVD. The GRUB configuration file will need to pass the install option to the kernel, which you can implement by replacing the contents of `/var/tmp/dvd/boot/grub/menu.lst` with the following:

```
default=0
timeout=15
title Custom Jumpstart
        kernel /boot/multiboot kernel/unix - install -B install_media=cdrom
        module /boot/x86.miniroot
```

If you are booting a system that uses a serial port as its console, then you should consider the following issues concerning output to the serial port as a console:

- Where GRUB is going to display its output.

You handle this at the BIOS level with BIOS console redirection, or with the `serial` and `terminal` keywords in the GRUB configuration file (`/boot/grub/menu.lst`).

- Where the Solaris kernel is going to display its output.

  You handle this through a `-B console=ttyX` option to the kernel line in `menu.lst`, or via the `eeprom 'console'` setting.

For example, if your console is on serial port A, and you have BIOS redirection, you can change the kernel line above to read:

```
kernel /boot/multiboot kernel/unix - install -B install_media=cdrom,console=ttya
```

If you do not have BIOS console redirection, then add the following under the `timeout` line:

```
serial --unit=0 --speed=9600
terminal serial
```

If you plan to use your customized DVD on systems with a number of different console configurations, it is possible to create a GRUB configuration that will adapt. That way, you do not need to create different DVDs for systems that have different console devices. The following `menu.lst` file can be used on systems that have either a kvm (keyboard, video, mouse) console or a serial console:

```
default=0
timeout=30
serial --unit=0 --speed=9600
terminal --timeout=10 console serial

title Custom Jumpstart (normal console)
        kernel /boot/multiboot kernel/unix - install -B install_media=cdrom
        module /boot/x86.miniroot

title Custom Jumpstart (serial console)
kernel /boot/multiboot kernel/unix - install -B install_media=cdrom,console=ttya
module /boot/x86.miniroot
```

When a DVD is booted with this GRUB configuration, the message 'Press any key to continue' will be displayed on all the devices listed in the `menu.lst` terminal entry—every second until the terminal timeout value is exceeded or a key is pressed.

- If a key is pressed, GRUB will select that device as the console and display the menu there.
- If no keypress is received within the terminal timeout value, the first device specified in the terminal entry will be chosen by default.

Once the console device is selected for GRUB, it is up to the user to select the correct menu item that corresponds to that console device for the Solaris kernel.

- If the wrong menu item is selected, then the kernel will most likely hang while trying to display its output to the non-console device.
- If no menu item is selected in the amount of time specified by the timeout value (30 seconds in the example above), then the menu item defined by the default entry will be used.

If you were to allow the DVD with the `menu.lst` above to boot without interaction, then the kvm console device would be chosen for both GRUB and the Solaris kernel. This could easily be changed to use the serial port as the default console for both.

For more information on GRUB configuration, refer to the online documentation at:

http://www.gnu.org/software/grub/manual/html_node/Configuration.html#Configuration

Step 4: Recreate the ISO Image

When all customizations are complete in the miniroot and `/var/tmp/dvd`, you simply run `mkisofs` to recreate the ISO image. The `mkisofs` command has the following syntax:

```
# /usr/bin/mkisofs -d -D -J -l -r -U \
    -relaxed-filenames \
    -b boot/grub/stage2_eltorito \
    -no-emul-boot  \
    -boot-load-size 4  \
    -boot-info-table   \
    -c .catalog   \
    -V "my_volume_name"  \
    -o output.iso  \
    /var/tmp/dvd
```

Use of the `-V` option is optional. If you use the `-V` option, then this will be the volume name seen when the DVD is mounted (such as `/cdrom/my_volume_name`) by vold in the Solaris OS. Once you have the `output.iso` file, you can then burn it to the DVD.

Conclusion

Once the ISO image file is created, there are numerous applications on various operating systems that can be used to perform the actual data transfer to a writable DVD. The key is to treat the ISO image file like any other. One pitfall that can happen with this is to create a DVD that had a single file in its directory hierarchy (the ISO image file). When burning the DVD, ensure that the software you use treats the ISO image file as such, and does not try to create its own directory structure. Use the `-i` option to the Solaris OS `cdrw` command to do this correctly.

This article provided a framework for implementing miniroot customizations for the Solaris OS for x86 Platforms DVDs. The examples in this article were used for instructional purposes. However, the concepts can be applied to whatever miniroot customizations you need to make.

References

- Advanced Configuration & Power Interface (ACPI) Specification

  http://www.acpi.info

- Building a Bootable DVD to Deploy a Solaris Flash Archive (April 2004) - John S. Howard

  http://www.sun.com/blueprints/0404/817-6991.pdf

- Creating a Customized Solaris on x86 Boot CD/DVD (December 2005) - John Cecere and Dana Fagerstrom

  http://www.sun.com/blueprints/1205/819-3731.pdf

- Customizing JumpStart Framework for Installation and Recovery (August 2002) - John S. Howard and Alex Noordergraaf

  http://www.sun.com/blueprints/0802/816-7587-10.pdf

- Download URL for large miniroot procedure

  http://javashoplm.sun.com/ECom/docs/Welcome.jsp?StoreId=8&PartDetailId=GRUB-SOL-x86-G-F&TransactionId=try

- El Torito Bootable CD-ROM Format Specification Version 1.0. Phoenix Technologies Inc., January 25, 1995.

  http://www.phoenix.com/NR/rdonlyres/98D3219C-9CC9-4DF5-B496-A286D893E36A/0/specscdrom.pdf

- fdisk partition types

  http://www.win.tue.nl/~aeb/partitions/partition_types-1.html

- GRUB and the Solaris 10 1/06 OS: The New Bootloader for x86 Platforms (December 2005) - Shudong Zhou and Jan Setje-Eilers

  http://www.sun.com/bigadmin/features/articles/grub_boot_solaris.html

- GRUB online documentation

  http://www.gnu.org/software/grub/manual/grub.html

- Opensolaris

  http://www.opensolaris.org

- Performing Network Installations Without a Local Boot Server (May 2004) - John S. Howard

  http://www.sun.com/blueprints/0504/817-7288.pdf

- Solaris 10 Installation Documentation

  http://docs.sun.com/app/docs/coll/1236.1

About the Author

John Cecere started his career in computers with the United States Air Force, where he worked on Sperry Univac mainframes. He took this experience to the civilian sector, where he perform similar job functions at Bell Communications Research. There he first encountered Unix, and he went on to be a system administrator for five years. He joined Sun in 1997 as a System Support Engineer and subsequently a Regional SSE. He is currently involved in several projects writing internal tools for Sun Services.

Acknowledgements

The author would like to thank Shudong Zhou and Jan Setje-Eilers from the Solaris on x86 Kernel Engineering Group for taking time out of their busy schedules to answer questions about the Solaris implementation of GRUB. These two have written a related article on the boot process itself for BigAdmin:

http://www.sun.com/bigadmin/features/articles/grub_boot_solaris.html

Thanks to Dana Fagerstrom who helped me write the first article, from which some of the information here was taken.

Ordering Sun Documents

The SunDocs^{SM} program provides more than 250 manuals from Sun Microsystems, Inc. If you live in the United States, Canada, Europe, or Japan, you can purchase documentation sets or individual manuals through this program.

Accessing Sun Documentation Online

The `docs.sun.com` Web site enables you to access Sun technical documentation online. You can browse the `docs.sun.com` archive or search for a specific book title or subject at `http://docs.sun.com/`.

To reference Sun BluePrints OnLine articles, visit the Sun BluePrints OnLine Web site at: `http://www.sun.com/blueprints/online.html`