Sun Microsystems Project #2

Porting Project Indiana from x86 to SPARC

Summer Field Session 2008

Keith Mitchell

Ricky Walker

Chris Walsh

Abstract

Sun Microsystems is currently developing an open source operating system called OpenSolaris, more specifically - Project Indiana. Currently, Project Indiana will only run on the x86 machine architecture, however Sun has expressed interest in developing a version that will run on the SPARC machine architecture. The goal of this project has been to develop a version of Project Indiana that will run on the SPARC machine architecture. This paper a brief introduction to the background information related to the project, a description of the completion strategy for the project, and a report on the final status of the project.

1 - Introduction

Sun Microsystems is currently developing OpenSolaris, an open source version of the Solaris operating system. The development project has been given the name Project Indiana, and currently only runs on the x86 machine architecture. Sun Microsystems has expressed interest in making Project Indiana run on the SPARC machine architecture. Thus, it has been the goal of this project to create a version of OpenSolaris that will run on the SPARC architecture.

Due to the size of the project and the short amount of time allotted, completion was not expected. The depth of the project was not well known in advance. In addition, because of the complicated nature of the project, a steep learning curve was expected in order to learn what was necessary for the project. In light of this, the plan towards completion was set as series of benchmarks. By finishing and documenting each benchmark before moving on, we ensured our client, Sun Microsystems, a solid pickup point from which to resume.

The first major benchmark for the project was to gain a thorough understanding of the boot process of a computer, particularly the boot process of OpenSolaris on both x86 and SPARC machine architecture. After understanding the boot process of the operating system, the second benchmark was to determine a plan of attack for the project and obtain a working environment for developing a solution. For completing the third benchmark, it became possible to split the workload into individual and parallel tracks.

Three tracks were scoped out as an efficient method towards completing the project. The first track consisted of creating a package repository that contained only the necessary components needed for liveCD booting and preparing a tool called Distribution Constructor (DC) to speed along the handcrafted test cycles. The second track consisted of recompiling and incorporating existing x86 utilities needed for liveCD booting over to the SPARC distribution, and cutting out any unneeded binaries and libraries from the existing SPARC distribution. The third and final track consisted of updating the Distribution Constructor tool to further speed our test cycles by automating the construction of our otherwise handcrafted test components; this involved accounting for the differences in compression and packaging of x86 and SPARC's respective boot environments.

After completing those tracks, the final benchmark was to successfully boot the machine to the point where it would call and execute the utilities (which would load the rest of the OS) that were created in track two.

2 - The Computer Boot Process

The first benchmark of the project was to obtain a basic knowledge of the boot process of a computer. While not all information pertaining to the boot process was directly applicable to our project, an understanding of the process of booting was very useful for understanding the process of what was occurring when starting the computer.

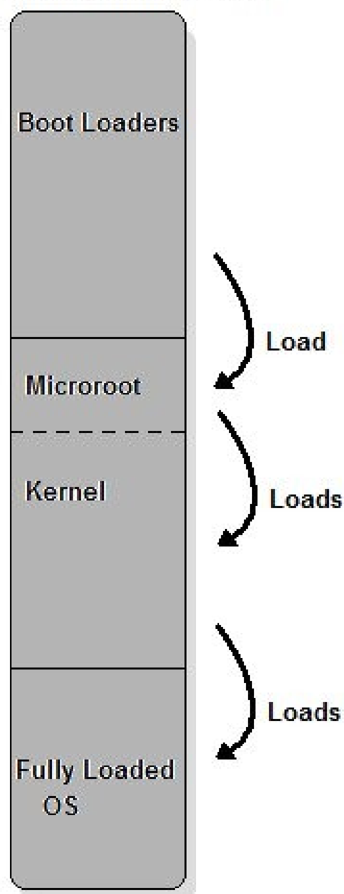

Figure 1

The process of booting a computer can be tricky in some ways. The ultimate goal is that you would like to have the operating system completely in memory. However, in order to load the operating system into memory, you need an operating system to load the operating system into memory, thus creating a circular problem. The solution to the problem comes in the form of creating several stages of boot loaders, written with the task of starting the operating system as illustrated in Figure 1.

The first level boot loader is an extremely small program implemented in hardware, with the task of recognizing hardware attached to the computer and loading the second level boot loader. Examples of first level boot loaders are BIOS on x86 and Open Boot Prom (OBP) on SPARC. The second level boot loader is a larger program that mounts the root file system, selects an operating system,

and then loads the kernel into memory. The second level boot loader is the only part of the boot process that is required to understand the file system format of the computer (whether the computer is using UFS, ZFS, NTFS, etc). Two examples of second level boot loaders are GRUB (for x86 systems) and NewBoot (for Solaris based Operating Systems). Once the kernel has been loaded into memory, it proceeds to load final modules required for the operating system to run, thus completing the booting process.

In the older versions of the Solaris boot sequence, the various boot stages would communicate between each other in a symbiotic 'dance' of pass-though function calls and memory map synchronizations. Fortunately the old booting sequence is being phased out, and thus knowing its exact implementation will not be necessary for this project. This is not the case for the modern booting process used in OpenSolaris. Now the boot stages are executed in a strict serial fashion and the structure of each stage allows them to execute independent of the others.

The boot process for OpenSolaris on x86 and SPARC are very similar, with the exception of which bootloaders it makes use of. Within both architechtures, the second level boot loader loads a microroot into memory, which contains the code necessary to implement a minimally functional user-land. From within this microroot, the rest of the file system is mounted and the kernel is loaded. The key to the completion of this project is the creation of the microroot.

3 - The Development Environment

The development environment for this project consists of three very important pieces: a Project Nevada workstation, an IPS repository, and a Distribution Constructor.

The first piece of the development environment for this project is the biweekly developer's release of Solaris, named project Nevada. The project Nevada workstation is an important part of the development environment because it provides a development environment on a machine with a SPARC architecture. Project Nevada can be thought of as a mix between Solaris OS and OpenSolaris. On the one hand, much of project Nevada is open source and exactly the same as OpenSolaris, but on the other hand some of project Nevada is closed source and has features that OpenSolaris does not. The most important feature of project Nevada from the standpoint of this project is that it contains the code required to install on a SPARC machine, thus giving us a stable platform to work on when dealing with SPARC machines.

In addition to providing an operating system, our Nevada Workstation contains two hard drives, allowing the microroot to be placed on a separate hard drive, which can then be used to boot from for testing.

The second piece of the development environment for the project is the IPS repository. The IPS repository is a repository containing all packages that are contained in the OpenSolaris operating system. It is similar to most repositories, except it automatically determines dependencies between packages at the code level. This feature makes it a valuable resource when creating an OpenSolaris distribution, because it enables us to chose what packages should or should not be included in our final distribution with ease.

The final piece of the development environment (and arguably the most important) is the Distribution Constructor (DC). The purpose of the Distribution Constructor is to create an ISO image which can be used to boot a computer from a live DVD. See Figure 2 for more reference.

The DC has three high level stages. The first stage is to pull packages from the IPS repository. This list of packages is merely saved as a text file, and the DC manages the packages as necessary. After pulling the appropriate packages, the DC proceeds to compress the packages to a size that will fit. After compressing the packages, the final step that the DC performs is the creation of the ISO image, which can then be burned to a CD, DVD or USB Drive, depending on the final size.

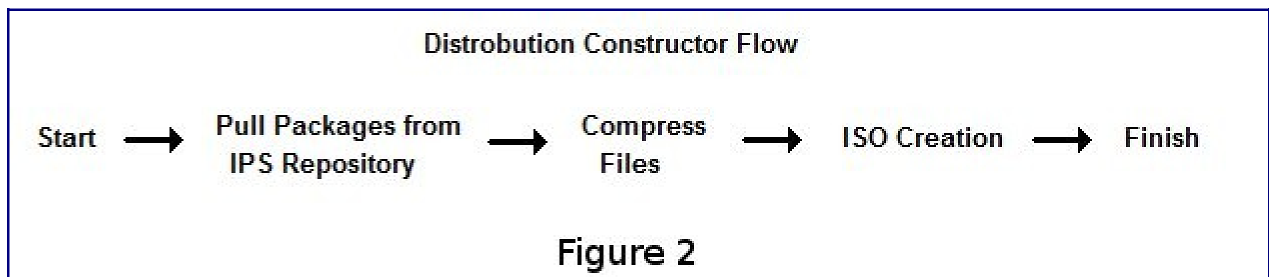

Figure 2

By debugging and running the DC on a SPARC machine running project Nevada, it becomes possible to create an ISO image that will run a live version of OpenSolaris on a SPARC machine.

4 - Project Completion Strategy

4.1 - Track One: The IPS Repository

The first track involved creating the IPS repository, and was a relatively simple task. In order for a machine to host a repository, it needs to have Sun Studio 11 and the "onbld" tools installed on it. At the time of this project, installation of Sun Studio 11 required use of JRE 1.5. After the installation of both of these packages, it was necessary to ensure that the SUNWonbld package was also installed on the machine.

Starting the repository was a straightforward task. After installation of SUNWonbld, executing the command "/usr/lib/pkg.depotd -d /export/home/repo -p 10000" initialized the IPS server on the local machine, using port 10000. The IPS repository could also have been configured to run via Solaris SMF, enabling it to start

automatically on boot. However, for the purposes of this project, it was unnecessary to have the IPS repository running at all times.

The IPS server was then populated with packages needed in order to run the live CD. The list of packages used for this project was derived from the slim-install metapackage, found on the x86 live CD. In some cases, a package in the x86 CD was split into two subgroups for the SPARC version. For example, SUNWftp on x86 was found at SUNWftpu and SUNWftpr on SPARC, with the 'r' at the end of the split version standing for "root" and the 'u' standing for 'usr'. Some additional packages need to be recreated for SPARC, as they have no counterpart yet. These packages contain scripts and corresponding service manifests for initializing the boot from live CD, for example the "live-fs-root" script discussed in section 4.2.

Populating the repository was accomplished by sending various packages to the repository. The simplest way to send a package to the repository is by executing the command "pkgsend send [path-to-package-folder]". The pkgsend command defaults to sending to http://localhost:10000, but if the repository is elsewhere it can be indicated to the pkgsend command with the appropriate flag. A simple, though slow, method of getting all the packages to the IPS was to execute the command "pkgsend send *" from the Solaris_11/Product directory of the Nevada CD, which contained most of the required packages. Due to the differences between Project Nevada and Project Indiana, the repository became populated with extra and unnecessary packages. However, because the DC required an explicit list of packages to be included when creating a distribution, these extra packages did not matter.

4.2 - Track Two: Hacking The Microroot

There are several changes that needed to be done to some files in order for the microroot to perform its task. In some cases, scripts or lists of files needed to have explicit references changed from some-package-x86 to some-package-sparc. The "usr_microroot_files" and "var_microroot_files" contained several such references. In those situations, the file already existed on the sparc machine.

The script that required the most modification was "live-fs-root". During boot, the microroot was loaded into memory. However, it was blindly copied, and unaware of the device from which it came. The live-fs-root script made attempts to locate the device from which the microroot was loaded, and continue loading the kernel from that device. Some of the most important pieces of that functionality inside the script were the calls to the "listcd" and "listusb" utilities. These utilities provided the same function: they searched the hardware for devices that could be USB or CD devices. The live-fs-root took all matches returned from the listcd/listusb functions and checked the file structure to determine if that specific device was the one from which the microroot was loaded. In the case of matches returned by listusb, it checked if the device had a ufs

file structure, and if it did, it checked the files for solaris.zlib. In the case of matches returned by listcd, the script checked that the volume ID of the device existed and matched the value stored in the folder ".volumeid" on the microroot. The checks for the "ufs" filesystem type and for the volume ID were performed by ufs/fstyp and hsfs/fstyp commands, respectively.

Some changes were made to the script and the listcd utility for the sole purpose of shortening the test cycle. The listcd utility was modified to be less strict about the devices it searches for, so that it returns both CD drives and hard drives. This, of course, allows us to boot our test microroot's from hard drive. However, hsfs/fstyp can't return a volume ID on non-CD drives, and ufs/fstyp does not provide a volume ID. So for the purposes of testing, we ran the modified listcd function to retrieve potential hard disks, then used the usb portion of the live-fs-root script to find the correct hard drive.

4.3 - Track Three: The Distribution Constructor

For this project, a prototype version of the Distribution Constructor (DC) was utilized. This tool allowed the state of construction of the live CD to be saved, allowing the file structure to be modified, so that the DC could restart from the same stage in the construction. This utility was important to the success of the project, due to the fact that many alterations needed to be made to the DC for the project, and the long running time of the DC made it impractical to repeatedly restart the DC from the beginning.

The biggest change that was made to the DC was the script that converted the proto area file structure into the microroot file. On x86 architectures, the microroot is constructed of a compressed file structure containing uncompressed files. On SPARC systems, the microroot is an uncompressed filesystem containing compressed files. These individually compressed files are then marked to decompress on read. The benefit of using the SPARC approach is that it not only saves CD space using compression, but it also shrinks the memory footprint of the microroot on bootup. At this time, the SPARC version of the script is complete but not fully tested.

Secondary changes were made to the DC during the final week of the project. In the last week we had realized that our IPS packages we imported from our Nevada DVD were not running any of the post install scripts that their previous format (sysV) allowed. For further information specific to the post install scripts refer to Appendix B.2. To allow us to continue making test microroots, while part of the team worked on translating these post install scripts into a format IPS would understand, we created a version of the DC that allows the packages to come from a DVD in the sysV format. This involved invoking a new command to run the unpacking and installaltion of the packages. A limitation of this version of the DC is this new command does not gracefully handle missing paclage dependencies well. To get around this we had to make sure our important packages, that many held as dependencies, first.

5 - Final Progress

At the end of 6 weeks, we have several items of value. First, we have a miniroot that runs the live-fs-root script. Although the script fails, in getting to that point we learned that SMF functions properly for miniroots - and, presumably, microroots. We have a microroot constructed from sysV packages, and one constructed using IPS. Both microroots are at the same point: they panic upon failing to find /devices.

6 - Summary

Based on the achievements made during the course of this project, it can be said that significant progress has been made toward the goal of getting Project Indiana to run on a SPARC architecture. Due to the limited time span of the project, completion was not expected, however there have been numerous useful advances which Sun Microsystems can use as a stepping block towards their final goal of adding SPARC compatibility to OpenSolaris. It has been a challenging, but rewarding project, and the final product will be useful to those who follow in the footsteps of this project.

Appendix A - Glossary

Basic Input/Output System (BIOS) - The BIOS (unique to x86 machines) controls the early, initial booting of the machine. It is responsible for generating a device list and loading boot code from one of the devices, among other start-up tasks.

Distribution Constructor (DC) - The DC is a script written by Sun Microsystems that creates bootable live media from a supplied set of packages. In summary, it downloads requested packages, creates a directory based on files from the packages, and compresses the directory into three files that form the basis for live media. The three files include a compressed microroot, and two compressed, "extra" files that contain additional utilities not needed during the boot process.

Grand Unified Bootloader (GRUB) - On x86 machines, GRUB is a commonly used 2nd stage bootloader. Once loaded into memory by the initial bootloader, GRUB has enough awareness to load many different operating systems, or to initialize a third party 2nd stage bootloader to boot an unsupported operating system.

Image Packaging System (IPS) - Sun's method for upgrading and customizing a Solaris-based system. On the user end, the IPS allows one to update a package from an external package server ("repository"), or acquire additional, new packages for the system. From a developer standpoint, an IPS server can be used to upload new packages and modify existing packages for others to download.

Indiana - This project, led by Sun Microsystems, is attempting to create a fully open source operating system based off Solaris. It currently runs only on x86 hardware; the goal of this project has been to create a live CD based off Solaris that will run on SPARC, a big stepping stone to a complete SPARC port of Indiana.

Live Media (CD / DVD / USB) - Refers to a bootable media, such as a DVD, that will boot into a fully functioning operating system. This is different than installation media, which only provides enough of an operating system to decompress and copy files to a hard disk, which would then be able to fully boot into an OS.

Microroot - Smaller than a miniroot, a microroot is designed for live media usage. The microroot needs to be large enough to load the rest of the OS into RAM, but small enough to allow other programs to run on top of it, such as the rest of the kernel, and any programs the user runs when the OS is loaded.

Miniroot - The miniroot is used on installation CDs to load enough of an operating system to interact with the user and make decisions about how to install the OS.

Nevada - This version of the Solaris OS contains proprietary software (closed source). It currently runs on both x86 and SPARC hardware.

OpenBoot Prom (OBP) - The OBP is the SPARC version of BIOS. In general, it is a more robust low-level booter than BIOS. Like BIOS, it is responsible for listing devices and starting the boot sequence from one of the devices found.

Package - A combination of files and dependencies on other packages that adds some functionality to a Solaris OS. For example, the SUNWonbld package contains tools for compiling code, and has dependencies on utilities that the tools use.

Service Management Facility (SMF) - The SMF smartly manages services on Solaris operating systems. It tracks dependencies and failures, and attempts to restart services when possible.

SPARC - Scalable Processor ARChitecture, or SPARC, refers to processors and related hardware initially designed by Sun Microsystems. SPARC computers were the first to offer 64 bit processing.

UFS - Unix File System, or UFS, refers to one method that a hard disk may organize its data and files.

x86 - The most common processor and computer architecture on the consumer market, currently. The name comes from a series of Intel processors which had "86" in the name.

ZFS - The ZFS file system is an evolution of UFS, developed by Sun Microsystems, to further abstract the file system from the user. Additionally, ZFS provides additional tools, such as "snapshots," or copies of a portion of the hard drive at a specific point in time, similar to a version control system such as CVS or SVN.

Appendix B - "Cut Corners, Loose Ends, and Miscellaneous Menucia"

Full completion of our project could take over a year. In order to progress as far as possible in the limited time available, we cut some corners in order to skirt problems that would take much time to solve for minimal reward.

B.1 - Compression

The x86 live CD's microroot is a directory structure of core files needed to boot. In order to save space, the entire directory is compressed. For our SPARC port, the preferred design of the microroot is a directory structure containing compressed files. The Solaris command used for compressing the files is "fiocompress." Regardless of the method, compression is important to a live CD microroot, because there is a limited amount of space on the CD, and the system that runs the CD may have limited available RAM. Additionally, the CD cannot assume there is hard disk space available for virtual memory use, because the existing drive(s) could be in filesystem formats that the CD does not understand.

When writing a script to perform the compression for us, we ran into permissions errors. That is, the compressed file no longer held its previous permissions. This turned any executable binary into a lifeless lump of ones and zeros. Further more, the "chmod" command, which changes permissions, failed to execute on any of the compressed files. After a short time, we decided that overcoming the permissions issue was irrelevant to the immediate needs of the project. Our development microroot boots off a hard drive, and our SPARC computers each had 8 gigabytes of RAM - more than enough to account for decompressed files.

The immediate issue of this problem is that during the the finalize stage of the DC, binaries from within the microroot are called to execute upon itself. The DC gets very unhappy when its calling files that the operating system refuses to execute. A quick fix to this problem would be to complete the compression at a later stage. We could let the DC create and finalize a temporary microroot, and then right before it commits the files to an ISO, we would compress and move the files into a final microroot which would then be commited into the ISO. The more unknown aspect that needs to be tested to confirm that this a problem of larger importance is if the lack of permissions extend to the very early stages of boot. Does OBP or the microroot care about this? If we had more time would have delved deeper into this unknown.

B.2 - IPS Actions

The Solaris and OpenSolaris OS's are migrating their packaging and system update methods to IPS. The old system used "postinstall" scripts to attach drivers and related files to the installed OS. The IPS system performs the same tasks using "action commands." Because the migration is still in progress, none of our SPARC packages utilized the IPS actions. As a result, our initial microroot constructions did not have correct driver listings.

The long term solution to this problem will be to convert the postinstall scripts of packages used into IPS actions. This is a tedious task, prone to error, requiring a knowledge of what tasks the postinstall scripts may be trying to do. While we did spend some time doing initial work on these conversions, they remain untested, and it is very possible that some actions were missed. As a short term solution, we copied pristine versions of the old packages from the Nevada install DVD onto our hard drive. This allowed us to construct a microroot - though it may have caused some of our other issues, as the old system had problems of its own.

B.3 - Booting from HD for our Test Cycles

As mentioned, our test cycle involved booting from a second hard drive, instead of burning files to a CD. This allowed us to shorten our test cycle considerably, saving the time of creating iso's, burning them, rebooting, etc. However, booting from HD brought additional challenges of its own. In particular, we had to manually install boot blocks and boot archives into the secondary hard drive, as well as the microroot. Additionally, as mentioned previously in the last paragraph of section 4.2, some of the live scripts and executables needed to be modified slightly in order to allow them to perform as they would from CD.

B.4 - The Latest Microroot

While hacking the microroot together during the last few days of the project, we didn't have much time at all to troubleshoot some issues. In order to speed up the testing cycle, and to see how far we could get on the final day, we began copying select files directly from the installer miniroot into our constructed microroot. Specifically, we copied etc/driver_aliases, etc/driver_classes, and etc/name_to_major. The general purpose of those files is to map device drivers in the kernel.

B.5 - Build Numbers

All packages in our IPS repo were taken from a Nevada build 88 DVD.
All sysV packages used with our special version of the DC and sysV microroot were from a Nevada build 90 DVD

Anything pertaining to, or taken from a miniroot was from a Nevada build 90 miniroot.