Preface and Introduction

The *Solaris® Porting Guide* has been written to help you gain insight into the Solaris 2.x environment and its related migration strategy. This book contains an overview of key technologies incorporated into Solaris 2.x.

It is written both for software developers as well as for key decision makers in any company interested in better understanding Solaris for SPARC® and x86.

How This Book Is Organized

Although this book can be read in sequence cover-to-cover, it is designed so you can skip to the chapters you need to use. This book is intended to serve both as an overview and as a reference document.

The **Introduction** should be read by all developers and key decision makers in your organization. It is intended as both a technical and marketing overview of Solaris and SVR4 and includes a migration checklist to Solaris 2.x.

The remainder of this book is dedicated to providing the required in-depth technical information to facilitate your migration efforts and understanding of the environment.

Chapter 1, Dynamic Linking, provides an overview of dynamic linking concepts and describes the changes made in dynamic linking between SunOS™ 4.1.x and SunOS 5.x.

Chapter 2, ANSI C, outlines the migration path from common C in SunOS 4.1.x to Sun ANSI C in the Sun C 2.0.1 package and in Solaris 2.x.

Chapter 3, Software Packaging, provides an overview of packaging and the various package utilities available, along with the conventions that need to be followed for structuring, installing, updating, and removing software.

Chapter 4, OpenWindows, describes the OpenWindows™ Version 3.x product, including the OPEN LOOK® Window Manager, the OPEN LOOK Intrinsics Toolkit (OLIT), the OpenWindows Server (X11 Release 5/DPSbased), and Display PostScript®.

Chapter 5, Graphics, describes the major components of Solaris VISUAL, as well as migration issues of moving Solaris 1.x graphical applications that exist on SPARC platforms today onto the less platform-specific Solaris 2.x.

Chapter 6, Application Interoperability, discusses what you must provide in your applications to take full advantage of desktop integration technologies. It also encompasses how you should prepare your applications for the next generation of distributed, object-oriented facilities, which are inherently application and desktop spanning, such as Selections and Drag and Drop, the Classing Engine, the ToolTalk® Service, and Project DOE.

Chapter 7, Internationalization, describes how to design your software so that it can be ported to a specific language or region. The chapter also discusses how to include internationalization as part of the base design of your product so that software can be easily customized for each locale.

Chapter 8, Compliance with SVID3, describes how to develop source code that can be used in both the SunOS 4.1.x and the Solaris 2.x environments. SVID3 refers to the UNIX System V Interface Definition, Third Edition.

Chapter 9, Device Drivers, describes how to port your device drivers to the Solaris 2.x environment using DDI interfaces introduced with Solaris 2.x.

Chapter 10, Multiprocessing and Multithreading, describes concurrent programming concepts and practices that are suitable to Solaris threads.

Chapter 11, Realtime Processing, describes the realtime enhancements that have been incorporated into SunOS 5.x. The various components are shown along with sample programs.

Chapter 12, Networking, provides a brief history of distributed computing and discusses Sun's networking products. It also discusses the future of ONC™ (Open Network Computing).

Chapter 13, Interprocess Communication, focuses on the IPC™ (InterProcess Communication) services that are specifically intended for single-system use. The distributed services are mentioned for comparison and to aid you in selecting the right service to use.

Chapter 14, Architectural Differences, discusses the differences between running Solaris 2.x on the SPARC or x86 (specifically 386DX-class of microprocessor or higher) architectures.

Chapter 15, Standards Compliance, examines some of the ways that standardization can be dealt with and tries to draw some general rules on when and how standardization can be accomplished.

Chapter 16, Project DOE: Future Solaris Technologies, describes a major SunSoft initiative to bring the leverage of open systems to the application arena. DOE extends the Solaris model with the full range of capabilities, services and partnerships required by an enterprise-ready, distributed object environment. It provides a software framework for a wide spectrum of solutions and supports standards-based, interoperability within heterogeneous computing environments.

Appendix A, Comparison of cc Options, is a multipage table that compares `cc` options supported in Sun™ C (bundled compiler) in SunOS 4.1.x, in Sun ANSI C (SC 2.0.1) on SunOS 4.x, and in ANSI C (SC 2.0.1) on SunOS 5.x.

Appendix B, Sun C/Sun ANSI C Differences, describes the differences between the previous (K&R, or Kernighan & Ritchie) Sun C and Sun ANSI C, as implemented on SunOS 5.x.

Appendix C, OpenWindows Source Code, contains sample source code for OLIT and OpenWindows Version 4 server, including a tutorial on how to internationalize an OLIT program, text buffer utility functions for internationalized OLIT, man pages for new OLIT widgets, sample source for DDX-based CG3 framebuffer, and sample tablet device driver using the dynamically-loadable mechanism and X input extension.

Appendix D, SunOS 4.1.x—SunOS5.x Compatibility Diagrams, provides compatibility and migration diagrams (Venn diagrams) to assist you in transitioning from the SunOS 4.1.x-SV3.2 (SVID2) environment to the new SunOS 5.x (SVID3) environment.

How To Use This Book

Application developers interested in the Solaris software development environment will find relevant information in Chapters 1 through 3, which cover dynamic linking, ANSI C, and software packaging.

Developers writing GUI-based applications should refer to Chapters 4 through 6, which discuss OpenWindows, graphics, and application interoperability.

Developers requiring more detail about the Solaris and SVR4 UNIX systems, should refer to Chapters 7 through 15, which cover operating system details, including standards issues. Chapter 9 on device drivers is accompanied by source code for a tablet driver and an example of a DDX-based CG3 framebuffer (see Appendix C).

Developers whose applications have performance requirements will find the multiprocessing, multithreading, and realtime processing chapters to be of interest.

For developers porting from SPARC to x86, Chapter 14 will be useful.

Developers interested in future technologies, should read Chapter 16 on Project DOE.

Many of the chapters in this book are appropriate for the general SVR4 UNIX community, including the chapters on networking, IPC, standards, and others.

Source Code Diskette

This porting guide is accompanied by a diskette containing source code examples and assorted demos. The data is compressed on this diskette in tar file format. To extract the files, use the command

```
tar -xvf floppy-device
```

where `floppy-device` is the name of your floppy disk drive.

For example,

```
tar -xvf /dev/fd0a
```

Related Books

A wide scope of material is presented in this book. Rather than present one comprehensive book list, we have included an individual “Bibliography” at the end of each chapter, which contains a list of books, technical reports, and white papers that related to the topics discussed in that chapter.

Introduction

The goal of this book is to help you gain insight into the Solaris 2.x environment and migration strategy so that we may move forward in unison to realize the substantial new business opportunities as they present themselves. This chapter should be read by all developers and key decision makers in your organization. The remainder of this book is dedicated to providing the required in-depth technical information to facilitate your migration efforts and understanding of the environment.

The Evolution of the SVR4 Operating System

The success of System V Release 4.0 (SVR4) is best understood by looking back at 1987. At that time, Sun was doubling its revenues and installed basis annually. The UNIX market was growing at a very healthy rate, although not quite as rapidly as Sun. When compared to the installed base of PC and Mac users, UNIX represented only a small percentage of the entire computing industry. One major problem was application software: UNIX lacked both the breadth of applications and a common binary format to enable binary transportable or shrink-wrapped software. While there were many vendors offering UNIX implementations, each was just different enough that it caused third-party software developers to create and maintain a complex development environment to support the various systems. The weight of this added overhead forced third-party software developers to be extremely judicious about which platforms to support, supporting some but not others. This in effect hampered the growth of the UNIX market...until the arrival of SVR4.

Goals of SVR4

SVR4 was developed for three reasons:

- To unify the fragmented UNIX market by combining the three major derivatives--BSD/SunOS, System V, and Xenix--into one common environment.

  SVR4 brings together over 80% of the installed UNIX systems. This naturally unites both the commercial sector, which is predominantly System V or Xenix-based, and the scientific and technical markets, which are generally BSD/SunOS-based.

- To provide a foundation for open systems computing through a common environment.

  SVR4 includes standard facilities and interfaces that permit software developers to create a common source environment which alleviates the costly overhead of supporting multiple platforms. SVR4 also allows systems of dissimilar architecture to integrate cleanly into a heterogeneous computing environment.

- To enable the creation of binary transportable software, thus expanding the general opportunities for UNIX software suppliers.

Partners in SVR4 Development

The development of SVR4 was completed in the middle of 1990 by Sun and AT&T and their other development partners such as ICL and Interactive, who provided some of the early reference ports. Access to SVR4 was then opened to the entire membership of UNIX International (UI), an independent body composed of system vendors, software suppliers, and their customers.

The process for defining SVR4 was, for its time, very open, although not quite as open as today where UI, in cooperation with UNIX System Laboratories (USL), guides the technical evolution of the System V. End users, software developers, system administrators, resellers, and others were asked to list their system software issues and provide their feedback about what they expected from the next generation UNIX environment. The ultimate objective was to determine how to best enhance the usability and salability of UNIX. Based on this input, SVR4 took the best technologies from each of the three major UNIX platforms.

UNIX derivative	Contribution to SVR4
BSD/SunOS	C shell, sockets, ONC/NFS
SVR3	STREAMS, Korn shell, VFS, and internationalization
Xenix	general source code compatibility

Over 60 companies are delivering System V Release 4.0. SVR4 enjoys global and very broad-based support, running on a wide array of systems ranging from laptops to desktops to high-end multiprocessing systems. European vendors include ICL, Nixdorf/Siemens, and Olivetti. North America supporters include Sun, AT&T, NCR, and Pyramid. In Asia and the Pacific Rim, Sony, NEC, and Fujitsu all ship SVR4 to their customers.

Applications on SVR4

There are almost 3000 applications on SVR4, with many new applications being added each month. Within the existing base of SVR4 applications, approximately 750 run on SPARC machines. ICL, which sells an SVR4-based SPARC systems, has been proactively driving the concept of the ABI with its software development partners. As a result, the vast majority of the ICL-based applications will be immediately applicable to all Solaris 2.x-based SPARC machines. Most of the ICL applications do not currently exist on other SPARC platforms; this will enable relevant SPARC-based machines to move into segments of the markets that have previously been unaccessible.

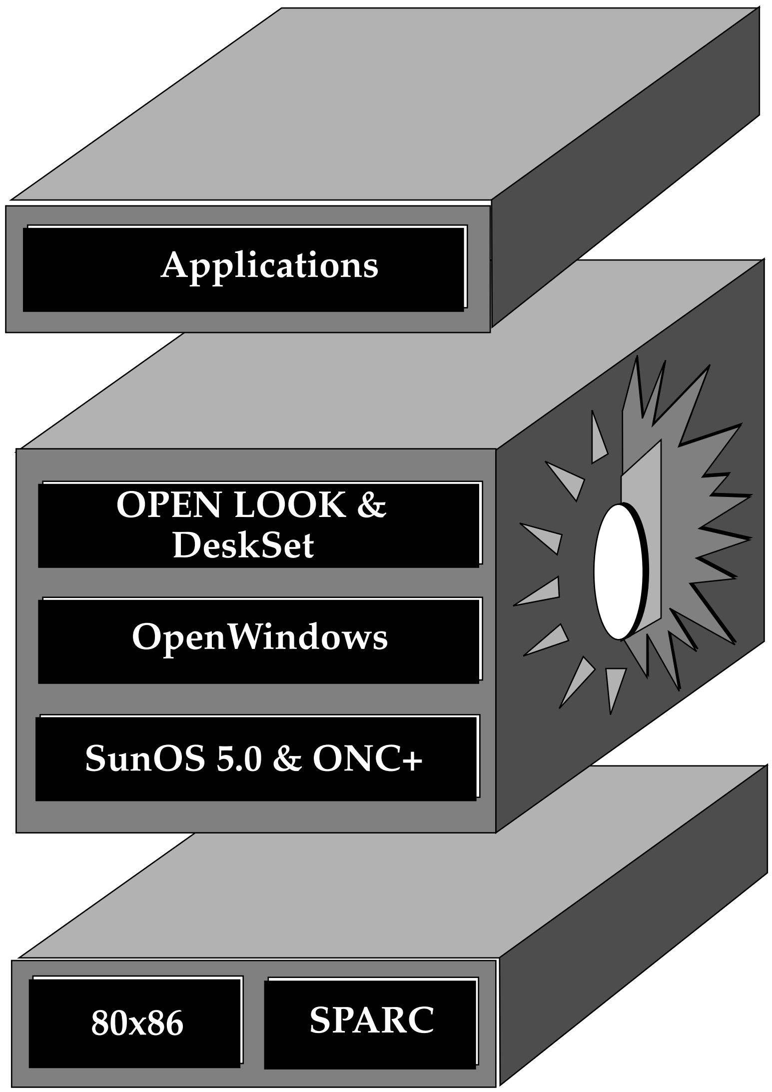
SunSoft and SVR4

SunSoft. the system software business unit of Sun Microsystems, Inc., officially opened for business in July 1991. The first order of business was to create an awareness of SunSoft’s company and products. Sun’s combined company and product launch, coupled with a developers conference introduced SunSoft to the world, provided a forum for Sun to articulate its strategic vision, and announced some of the relationships Sun had been developing over the preceding months. A key component of the announcement was the intent to port Solaris 2.x (which is based on SVR4) to both SPARC and Intel. To complement the Intel product, Sun announced strategic relationships with several important Intel-based platforms suppliers such as Dell, AST, and NetFrame.

At the announcement SunSoft had over 50 software developer partners announce their intention to port software to the Solaris 2.x environment. These vendors represent over 75 applications that address a wide range of market segments. They included database vendors such as Oracle, Ingres, Informix, and Sybase; design automation vendors such as EDS Unigraphics (previously McDonnell-Douglas), MCS, and SDRC; and productivity applications such as Lotus, WordPerfect, Island Graphics, and SAS. This is the basis of the software developer momentum behind Solaris 2.x.

What is Solaris?

Solaris is a full 32-bit distributed computing operating environment. The figure below shows the Solaris 2.x stack, indicating that the Solaris environment sits between the applications, at the top, and the hardware platform, at the bottom. There is nothing architecturally within Solaris that restricts it to a specific platform. Currently, Solaris 2.x is targeted to SPARC® and Intel®, two of the three volume platforms in the industry. (Macintosh® is the third platform.). The Solaris environment is truly portable, and SunSoft will consider supporting other high volume platforms as they emerge.



Goals of Solaris

Solaris was designed to solve three major problems confronting the UNIX industry:

- To address the performance gap.

This refers to the difference between what the hardware platform is theoretically capable of achieving and what the system software allows it to achieve.

Systems today are very powerful with a great deal of integrated technology. Unfortunately, the system software running on these systems has not evolved at the same rate. The result is that the full potential of these systems cannot be realized. Through Solaris' true multitasking, multi-user, symmetric multiprocessing, and multithreading capabilities, you can now unleash the real power of the underlying hardware.

- To address the shift in end-user needs.

  End-users are becoming more sophisticated. Their business environment is becoming more complex, and the information they required to solve problems are not centrally located but rather distributed throughout the entire corporation. Solaris provides the ideal environment for software developers to create distributed, client-server solutions. These applications allow end-users to more effectively solve their business problems by leveraging the power and resources of the enterprise computing environment.

- To support the heterogeneous computing environment.

  This computing model is here to stay, and end-users are demanding that they receive the maximum return on their information system technology investment. The technologies found in Solaris 2.x (some of which are licensed separately, such as Tooltalk and ONC/NFS) will help customers unite their heterogeneous computing environment and preserve their substantial investment.

What Solaris Provides for Software Developers

Solaris 2.x is truly a major step forward. It allows software developers, especially those who support multiple platforms, to remove some of the complexity from their development environment. Time to market can be accelerated. Another benefit is the expansion of current business opportunities. Today, software developers view the SPARC market not as a unified market, but rather as a series smaller, semi-independent markets. For example, they see Sun SPARC as a distinctly different market from Solbourne SPARC, which in turn is different from Toshiba SPARC. Though they all use the same processor and operating systems, they do not always run the same application binaries. The only way to know if they will run the same binary is to test it. The ABI allows software vendors to look at SPARC as a single market. This almost exactly emulates the DOS world where application developers generally do not distinguish between a Compaq PC and a Noname PC. To them a PC is a PC is a PC. Finally, Solaris 2.x provides a foundation for the future. All new enhancements and technologies will be targeted to the SVR4/Solaris 2.x environment.

Solaris 2.x is a complete environment that comes in one package, for one price, and all from the same supplier. It contains everything the developer needs: the core OS, networking, window system, and desktop tools.

Solaris' Added Value

Though Solaris 2.x is based on SVR4, SunSoft has added value at all levels of the Solaris environment. In general, incorporating or using the Solaris enhancements (or value added extensions) does not impact ABI compliance. In many cases, the SunSoft enhancements were made below the standard interfaces. As an example, Solaris 2.x includes a fully symmetric multiprocessing and multithreaded kernel that enables applications to potentially realize a performance improvement without any modification.   In other cases, SunSoft was required to provide functionality in excess of that provided via the standard interface. Facilities such as the Solaris application accessible thread interfaces that extend beyond the POSIX definition may provide improved functionality or performance. Use of these extensions is a "personal decision" based on the software developer's unique business plans and objectives.

International Standards

The Solaris 2.x environment supports international standards. For example, at the Foundation Technologies layer, Solaris 2.x is fully SVID 3 and XPG3 compliant. At the Developer Environment level, the window system is MIT X11R4 and ICCCM compliant. Standards permit software developers to leverage their development investment across a greater number of platforms and to earn a greater return without compromising functionality or robustness.

For end-users, standards mean there is improved integration of systems into the heterogeneous computing environment. ONC/NFS, the standard for heterogeneous networking and file sharing, is a core component of SVR4. ONC/NFS enables the tightest level of system inter-operation within the client/server computing model such that users can transparently access information on or from any combination of SVR4, SunOS and other ONC/NFS compliant systems.

Device Drivers and Interface Standards

Two important new features incorporated into SVR4 are DDI/DKI, the Device Driver Interface/Device Kernel Interface, and TPI/DLPI, the Transport Provider Interface/Data Link Provider Interface. In the past, drivers were always adversely impacted by the release of new hardware platforms and OS releases. Using these new interface standards, developers of custom drivers can shield their products from the subtleties of the hardware bus architecture and the internals of the kernel. Drivers written to DDI/DKI and TPI/DLPI are binary compatible across all SVR4 releases for a given platform and source code compatible across all SVR4 implementations; whether it is for SPARC, Intel, or another SVR4-based platform. Through these interfaces, driver vendors have achieved a greater degree of independence and flexibility; putting them a more equal footing with other application developers. This is a major and much welcomed step forward to creating shrink-wrapped driver software.

Multiple Binaries from a Single Source Tree

Probably one of the most important benefits of Solaris 2.x and SVR4 is that you can create multiple binaries from a single source tree. The overall effect of the one source/multiple binary strategy is to simplify the development environment.

To support the many UNIX variants, software developers currently maintain either a separate source tree for each platform or alternatively they may have a complex structure of `#ifdef` statements for conditional compilations. (Actually, there is a third alternative which is a combination of the two just mentioned: `#ifdef` statements followed by `#include` statements to bring in platform specific code.) Unfortunately, every time a source change is made, it either needs to percolate throughout the entire set of sources or requires a careful adjustment to and possibly increases the complexity of the `#ifdef` structure. This is unnecessarily burdensome on engineering, QA, manufacturing, customer support, and sales. With Solaris 2.x and SVR4, supported by the strict adherence to international standards, it is possible to collapse everything down to a single, simple source base with very few, if any, platform or OS-specific `#ifdef` statements. Such `#ifdef` statements permit the incorporation of value-added extensions, the use of which is supported by a compelling business case. This frees you from spending much of your valuable time and resources dealing with the mundane aspects of product engineering, such as source administration or porting and testing on new platforms or OS releases. Instead, you can spend more time developing new features and new products to enhance your competitive position, better meet your customers needs, and improve your revenue stream. Furthermore, if you adopt the migration strategy discussed later in more detail, you will have actually created a source base that can be used to address existing opportunities on SunOS 4.1.x as well as the rapidly emerging opportunities on SVR4 and Solaris 2.x.

The end result is a simplified development environment and greatly improved time to market.

Application Binary Interface (ABI)

With the introduction of Solaris 2.x and SVR4, we now gain the concept of an ABI, or Application Binary Interface. The ABI enables binary transportable or shrink-wrapped software for a given processor architecture. This means a single binary can be used to address the entire range of SPARC systems, while another binary is able to support all Intel-based systems (because it has an instruction set different from SPARC).

Solaris 1.0

SunOS 4.1.x is licensed, as part of Solaris 1.0, to the other SPARC hardware vendors. The SunOS provided by most SPARC vendors is substantially the same as that found on Sun systems so they can leverage off the large base of SPARC applications. Differences that might exist are generally attributed to the support of value-added features, such as Solbourne's MP or Mars Microsystems' DOS support, which usually do not impact binary compatibility. This has in effect created an informal ABI, permitting applications to run across all SPARC systems. The one drawback, however, is the only way to guarantee binary transportability is to test it. This testing is time consuming, costly, and in certain cases may not be justifiable given the limited revenue opportunities (relative to that provided by the Sun platforms).

Solaris 2.x

Solaris 2.x, on the other hand, with its formally defined SVR4 SPARC ABI, allows software developers, system vendors, customers to realize the many benefits of shrink-wrapped software. The ABI is defined by two documents:

1. The *System V Application Binary Interface*, or "Generic ABI", is applicable to all SVR4 implementations. The Generic ABI defines the system interfaces for compiled (binary) programs that are guaranteed to exist on any ABI compliant system. For example, it defines interfaces to facilities that print to the console or allocate a block of memory. The ABI also requires programs to be dynamically linked to specific system libraries. In this way, the system software—that is, what resides below the standard interfaces—is used to accommodate the architectural uniqueness of the hardware platform. This lets the physical differences between a SPARCstation 2 and an RDI laptop be isolated from the application developer and appear uniform in design. Those interfaces are verified to work, enabling application developers to feel confident that using these interfaces will not break compatibility. If an application statically links libraries, it is potentially bringing the incompatible baggage of one system to another. Finally, the ABI specifies a standard for software packaging and installation to simplify the distribution and installation of applications. The packaging scheme is flexible enough to allow multiple applications to be delivered on the same distribution media. This helps reduce manufacturing overhead and inventory costs. The distribution media is also defined within the ABI. For SPARC, the compliant media types are 3-1/2" floppy, 1/4" tape, and CDROM. If you use 1/2" tape you are automatically not ABI-compliant.

2. The processor supplement. There is a specific ABI for each processor which defines items such as fundamental data type, such as the length of an integer, and the possible contents of object file header fields. For most applications, the processor specific issues are transparent as they are handled by the compiler, linker, and system include files. On the other hand, some vendors developing CASE tools or compilers need to be aware of the differences between the various processor ABIs. In such cases, you will need to use `#ifdef` statements as you move from a SPARC to an Intel system because the architectures are inherently different. But, once again, from an application standpoint, the difference is not material.

PCs Versus SPARC Workstations

If you examine why the PC market has been tremendously successful, you can identify two main reasons. First, the standardization of the hardware platforms has enabled many players to enter the market and generate extremely high shipment volumes. Second, uniform hardware and software environments enabled the creation of shrink-wrapped software, which in turn attracted more applications to the platform. The ABI's ability to unite architecturally different systems to make them appear uniform will have a similarly positive effect on the growth of the UNIX market.

As a result of the ABI, UNIX software will eventually be delivered in the same manner as software in the PC world. You will be able to walk into your favorite computer store and purchase a shrink-wrapped copy of “Application Z” that will run on any SPARC-based system. As in the PC environment, you do not need to be concerned with the specific SPARC system just as you would not care about the specific brand of PC. Your primary concern is the processor architecture (SPARC or Intel 286, 386, etc.), the OS release (SVR4.X or DOS 5.0), and how much disk and memory you need to minimally run the application.

This means you have essentially removed the barriers to addressing new platforms. A key concern of software developers has been that the physical differences in systems require them to re-QA their applications on each platform, train support personnel, and obtain machines for their answer centers. With the ABI, if system vendors are ABI-compliant and the application is ABI-compliant, the application is guaranteed to run. The analogy in the PC world is if you visit Lotus or WordPerfect, you will not find a lab with 100, 60, or even 50 varieties of PC. They test on a couple and then say it should work for these general types of systems. It works because the system architectures are uniform. Once again, the ABI allows dissimilar system architectures to appear homogeneous to applications.

The ABI allows software developers to regain control of their development environment. Because the ABI is backward compatible, applications released today will run in the same manner in the future. You will release your products only when you have a sufficient suite of new features or critical mass of bug fixes, not because the system vendor has released a new platform or OS that has broken your application.

The system vendors are eager to support the ABI because it is in their best interest. First and foremost, it provides access to a wider array of application software due to the common APIs and binary transportability of applications. As system vendors move into new segments of the computing industry your applications will be brought along too, thereby realizing the possibility of additional, previously unrealized revenue opportunities. Furthermore, there is a real concern among system vendors that non-conformant systems will be application starved, as a substantial financial justification is required to port to a proprietary system. From now on, software developers will view systems as SPARC or Intel or whatever rather than unique systems from individual system vendors. Non-conformant systems will be qualified by the amount of business they will generate relative to entire segments of the industry. For example, the business generated by non-conformant SPARC system would be compared to the cumulative revenue stream of all conformant SPARC systems. The business case must be compelling for the third-party software developers to go through the additional effort. Finally, decommitting to the ABI means returning to the status quo; a fragmented UNIX market that requires third-party software developers to bear the burden of supporting many similar but incompatible UNIX-based operating systems, which in turn slows the overall UNIX market growth.

Solaris 2.x on SPARC and Intel

Solaris 2.x supports both the SPARC and Intel platforms. These are two distinct platforms that do not share the same binaries. Through a common source base, an application can be compiled once for Solaris 2.x on a SPARC platform and address the entire spectrum of systems from laptops to supercomputers using the same binary. Likewise, by recompiling for the Intel world, a single binary can address the large array of Intel systems. Software developers can now, using very little incremental effort, service an entire market that may have previously been unaccessible.

Solaris for x86

Many end-users are very interested in Solaris 2.x on Intel. First and foremost, Solaris 2.x enables end-users to integrate their PC more seamlessly into the heterogeneous computing environment. Until this point, end-users have had to rely on PC-NFS to link their PCs into the client-server computing model. PC-NFS was a stopgap measure that subordinated the PCs to other systems in the heterogeneous environment. With the advent of Solaris 2.x for Intel, PCs are elevated to an equal footing with other UNIX systems. Another important feature of Solaris 2.x is that the SPARC and Intel versions are exactly the same, feature-for-feature and bug-for-bug. When you see a PC running Solaris 2.x, it looks exactly like a workstation's version of Solaris. As the end-users' computing needs evolve, they can move up to workstations without impacting productivity because of their familiarity with the environment. This migration is simple and convenient.

Internationalized Solaris

The base environment of Solaris 2.x is fully internationalized and multi-byte character ready. Solaris 2.x includes all of the formats, collation routines, and messaging facilities to create internationalized applications. This is different from the past, where OS engineering completed the core development and then handed it off to another engineering group. This engineering organization would then take three to six months and render a wholly new OS capable of supporting the Asian language characters. The impact on software developers was significant, as they would need to create and QA distinct international versions of their products. With Solaris 2.x, you can use message catalogs instead of hard-coded strings, enabling one binary to support the entire worldwide market. In this situation, moving from one language to another requires the software developer to merely translate the message catalog to the new language rather than releasing different binary for each language. This is a great benefit for both software developers and end-users. Now, if a multinational company wants to standardize on a given application it will not need to wait three or more months for the Japanese version to appear; it will be made available almost coincidently with the support of other languages.

Multiprocessing and Multithreaded Kernel

Solaris 2.x has a full symmetric multiprocessing and multithreaded kernel. The three main benefits of this system software architecture are (1) enhanced application performance, (2) conformance to standards, and (3) access to a new systems.

The performance enhancements come in two stages:

1. The transparent performance improvements realized from the modifications to the kernel architecture. This is a true symmetric multiprocessing architecture, not a master/slave implementation, so every processor is capable of running the kernel and performing I/O operations.

   Whenever an application creates a child process by using fork that new child process, depending on CPU loads, can be spread to another processor. Similarly, when multiple instances of an application are initiated simultaneously, such as one instance of a ECAD application to do the design while another does the verification, they can run concurrently on different processors. Because the kernel is multithreaded, many processes can be "in the kernel" at the same time; one could be servicing an interrupt while another is servicing a system call. There is now a general improvement in parallelism which will benefit both uniprocessor and multiprocessor systems.

2. The second level of performance enhancement is obtained by incorporating the Solaris 2.x application accessible threads. Threads provide applications with multiple concurrent paths of execution within the same process space, as opposed to forking, which incurs a great deal of added overhead because it creates a whole new process. Incorporating these libraries routines may require some redesigning of the application.

The application accessible thread libraries will be a superset of the POSIX threads standard which is not yet finalized. To ensure there is no rework required on the part of the third-party software developers due to diverging SunSoft and POSIX threads standards and implementations, SunSoft will not release its threads libraries until the POSIX definition is completed.

This prepares you for the new generation of high performance, modular, multiprocessor systems coming down the line. The SPARCserver 600MP is just the start. You will want to verify your applications are prepared to take full advantage of the architectural features of these platforms.

Solaris Migration Path

Solaris is the base platform upon which all future development will be done. This includes Sun and Solaris, as well as all of the UNIX International and SPARC International vendors.

We expect to see new technologies and extensions delivered only on Solaris 2.x. You can expect facilities to support commercial computing, high availability, and multimedia. For the richest and most robust platform, Solaris 2.x is where you want to go. Moving to this environment enables you to reap many benefits, such as shrink-wrapped software, while at the same time enables you to make a solid investment in the future.

Migration Path Checklisting

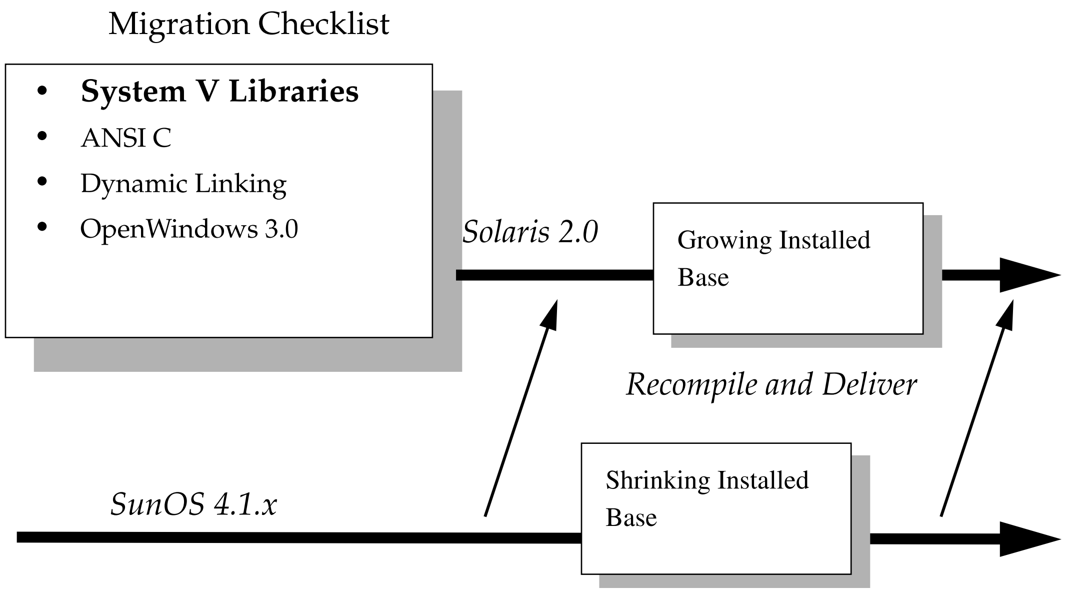
Software developers migrating to Solaris want a clean migration path; one that will allow them to move from where they are today to where they want to be. They want to ensure that while they are looking to the future, they are not cutting off their existing customer base. They want to leverage any development efforts to support both the new and the old environments. They also want to maximize their revenue opportunities and to recover their investment as quickly as possible. Finally, as a reality check, software developers want to verify that Sun itself is following this same approach. Sun is.

The migration strategy we developed based upon input from developers is called *checklisting*, and it really works.

Checklisting refers to the preparatory work that should be completed prior to moving to the Solaris 2.x environment. The checklist process entails

- moving to SunOS 4.1.x
- using the System V libraries and commands
- running under OpenWindows Version 3
- dynamically linking to external libraries
- incorporating ANSI C (if the application is written in the C language)

The following figure summarizes the Solaris 2.x migration strategy outlined in this chapter.



Checklisting has the following benefits:

1. You can build an experience base with the new SunSoft technology and familiarizes you with UNIX System V constructs.
2. Because it is a highly-leveraged effort, you can focus your resources by doing the "hard work" once and up-front rather than having separate development efforts for each Solaris environment. Also, because the checklist process renders a salable product, you can recover your development costs by selling into the still growing SunOS 4.1.x customer base.
3. You can proactively respond to the changing characteristics of the installed base. Over time, as customer demand for Solaris 2.x and SVR4 increases and according to your business plan or model, your applications can be moved effortlessly into the Solaris 2.x/SVR4 environment, whether written for the SPARC or Intel platform.

Why You Should Move to SunOS 4.1.x

There are both business and technical reasons for moving to SunOS 4.1.x.

On the technical side, because Solaris 1.0 is fully SVID 2 and 90% SVID 3 compliant (the System V Interface Definition for SVR4), most, if not all of the migration work can be done now in anticipation of the Solaris 2.x transition. Solaris 1.0 is the best environment for developing SVR4/Solaris 2.x-ready software. It hosts a wide array of development tools, such as compilers, source and binary level debuggers, and profilers, in addition to the applications required to round out the end-user environment, such as databases, word processors, and CAD tools. However, because Solaris 1.0 is not fully SVID 3 compliant, some work must be deferred until the Solaris 2.x timeframe. Areas affected are device and network drivers because the DDI/DKI and TPI/DLPI are not included in 1.0. Also, work related to the object file format may not be completed in advance because Solaris 1.0 uses the `a.out` format while Solaris 2.x uses ELF (Executable and Linking Format).

On the business side, there is a very large and growing installed base on SunOS 4.1.x. Sun is proactively pulling the customer base to this environment by offering both hardware and software products that only run on SunOS 4.1.x, such as OpenWindows Version 3.0 and SPARCstation 2s. The creation of a common source base, one that services both SunOS 4.1.x and Solaris 2.x, allows you to proactively move to the new environment without compromising the support provided to your existing customer base.

Why You Should Use the System V Libraries

In SunOS 4.1.x, BSD is the primary operating environment and System V is the secondary environment. In SVR4 and Solaris 2.x, the standard is System V and the optional environment is BSD. Converting to the System V libraries is essential because these are the system interfaces specified in the ABI. Recompiling your BSD-based application as it is today is possible; however, keep in mind that to gain the true benefits of the Solaris 2.x environment, such as shrink-wrapped software, you must be ABI-compliant and the BSD interfaces fall outside the scope of the ABI.

To assist in the adoption of the System V interfaces, SunSoft has developed a special tool call Pipeline/svmt. The two objectives of Pipeline/svmt are to identify the scope of the required porting work and to facilitate the migration process. The Pipeline Tool is extremely valuable because you can use it as an analysis tool to aid in migration resource and schedule planning. The summary report generated by Pipeline/svmt helps eliminate the common misconception that moving to Solaris 2.x is very time-consuming and resource-intensive. Pipeline/svmt is not a code converter, but instead a source code scanner that highlights the potentially problematic areas and offers a solution to resolve the problem. Pipeline/svmt includes a series of switches to assist in achieving, for example, SVID2, SVID3, and ABI compliance. Pipeline/svmt can be used as either a “batch” mode tool or as an interactive facility that incorporates an OPEN LOOK interface.

Why Your Software Should Run Under OpenWindows Version 3

OpenWindows Version 3 is *the* window system for Solaris 2.x. There is no more SunView. SunView applications will need to move to OpenWindows now. The programmatic interfaces for OpenWindows Version 3, whether on Solaris 1.0 or Solaris 2.x, are exactly the same. There is no need to be concerned about using OLIT on Solaris 1.0 and then having to modify your GUI code once again when you move to OLIT under Solaris 2.x.

OpenWindows Version 3, in addition to offering enhanced productivity tools such as the audio tool and multimedia mail, incorporates some important new technologies. SunSoft has now formalized the cut-and-paste and drag-and-drop protocols so applications may be better integrated into the OpenWindows desktop and operate better with other entities in the environment. You will also find ToolTalk™, SunSoft's new object messaging and multicast facility, which extends the level of communication between applications. ToolTalk can be used to scope the communication around one of two resources: a session which restricts communication to entities running on the same machine, or a file which allows applications to inter-operate across the network. There are processes in place for defining standard messages for given market segments. Currently, there are CASE and EDA message alliances. The goal is to allow all applications within a given space to be able to inter-operate.

As a side note, ToolTalk is the first step in the path to the object-oriented environment defined by SunSoft's Project DOE (Distributed Objects Everywhere). DOE encompasses the DOMF, distributed object management facility, the joint Sun/HP submission to the Object Management Group (OMG). DOE will be a superset of the OMG specification. Needless to say, using ToolTalk today brings the object-oriented environment of the future into closer reach.

Dynamic Linking to External Libraries

Dynamic linking is the next step in the migration checklist. As discussed earlier, the ABI's architecture allows a clean separation of the application from the underlying environment. The ABI guarantees specific libraries exist on all compliant machines and ensures the integrity of the standard interfaces. Because the uniqueness of the platform is accommodated below these interfaces, if you do not dynamically link, you potentially carry the incompatible baggage of one system to another. There are four libraries to which applications must dynamically link to be ABI compliant. These libraries are `libc`, `libsys`, which is also used by `libc`, `libx`, and `libnsl`—the networking services. All other libraries can be either statically or dynamically linked, depending on the form the library is provided. You are strongly urged to dynamically link wherever possible as it creates smaller binaries which use less memory and therefore swap less; this may improve system and network performance.

Why You Should Use ANSI C

ANSI C is an important part of the standards thrust inherent in the Solaris 2.x migration. ANSI C removes many of the ambiguities found in the current C compiler specification and implementations, such as poor type checking, which have in the past hindered application portability.

Realizing that the conversion to ANSI C could be one of the most time consuming aspects of the migration, the ANSI C compiler includes a number switches to accommodate various degrees of ANSI C compliance. Based on the switch you select, the compiler will accept anything from common C constructs only through ANSI C semantics only. In this way, the compiler itself can actually be used as a lint tool to help you convert your code to conform to the ANSI standard.

Once you have gone through these five migration steps, you will have created a common source base which lets you compile once for the SunOS 4.1.x environment and once for Solaris 2.x. You will be in an excellent position to respond to what we believe, based on our discussions with customers and third-party software developers alike, a rapid migration to Solaris 2.x.

The Future of Solaris 2.x

Solaris 2.x and SVR4 will consolidate the existing installed base of UNIX systems, while at the same time open up a host of new business opportunities. Through key technologies, such as the ABI, DDI/DKI, and TPI/DLPI, application developers and hardware vendors can leverage their development investments and obtain greater flexibility. By the same token, end-users can leverage their hardware and software investments and can purchase better overall solutions to their business problems.