ORACLE
SOLARIS

An Oracle White Paper
August 2011

HP-UX to Oracle Solaris Porting Guide

Getting Started on the Move to Oracle Solaris

ORACLE

Chapter 1 Introduction 1
Oracle Solaris 1
The Advantages of Porting to Oracle Solaris 1
Chapter 2 The Porting Process 3
Infrastructure and Application Porting Assessment 3
Build Environment Deployment 4
Data Integration 4
Source Code Porting 5
Application Verification 5
Commercial Applications and Third-Party Products 5
Chapter 3 Operating System Considerations 6
Processor Endianness 6
Data Alignment 6
Read/Write Structures 7
Storage Order and Alignment 7
64-Bit Data Models 7
Chapter 4 Runtime Environment 10
Environment Variables 10
Permissions 10
Process Resource and Runtime Limits 10
Application Programming Interfaces 12
System Libraries 12
Shells and Utilities 12
Scripts 13
Chapter 5 Devices 14
Device Naming Conventions 14
Device Driver Interface/Driver Kernel Interface 15
Best Practices for Porting Device Drivers 16

Chapter 6 Development Environment 18
Oracle Solaris Studio Components 19
Java Programming Tools 22
Developing Applications 23
Building Applications 31
Debugging Applications 37
Optimizing Applications 39

Chapter 7 Threads and Multiprocessing 49
Threading Models 49

Chapter 8 Distributing Applications 52
Preparing an Application for Packaging 52
Building a Package 53

Chapter 9 Running Applications 54
Oracle Solaris Service Management Facility 54
Continued Support for .rc Scripts 58

Chapter 10 File Systems and Data 59
File Systems 59
Data Transformation 63

Chapter 11 Clustering 65
Oracle Real Applications Cluster 65
Oracle Solaris Cluster 65
Differences Between HP Serviceguard and Oracle Solaris Cluster 69

Chapter 12 Building Secure Applications 71
Security Interfaces for Developers 71

Chapter 13 Internationalization and Localization 78
Overview 78
Encoding Methods 79

Input Methods 81
Codeset Converters 83
Locales 84
Message Catalogs 84
X and Motif Applications 84

Appendix A C Library Mapping 85

Appendix B API Differences 86

Appendix C Summary of Supported Locales 87

Appendix D Privileges Interfaces 93

Appendix E Cryptographic Functions 94

Appendix F Command Comparison Summary 96

Appendix G Resources 99

Appendix H Glossary 101

Chapter 1 Introduction

Today many IT organizations recognize the inability of legacy systems to respond to growing service and application demands. For many, HP systems running the HP-UX 11i operating system — particularly those based on Intel® Itanium® processors — are failing to keep pace. With the future of the platform uncertain, independent software vendor (ISV) support for these systems is waning. As enterprises look for alternatives, Oracle's SPARC and x86 systems running the Oracle® Solaris operating system emerge as an obvious, and safe, choice for porting critical business applications. HP-UX and Oracle Solaris share a common UNIX history—and are more alike than they are different—making application porting a straightforward task.

Once applications are ported to Oracle Solaris, developers can focus on application enhancements rather than worrying about adherence to operating system updates. A binary application guarantee ensures that applications ported to the Oracle Solaris Application Binary Interface will run without modification on all updates to Oracle Solaris as well as later releases of the operating system. In addition, source code compatibility between platforms ensures applications ported to SPARC systems can be recompiled easily on x86 systems and vice versa.

This guide serves as a porting roadmap, providing insight into the issues and best practices to consider when porting applications from HP-UX 11i v3 to the Oracle Solaris operating system. Included is an overview of the differences between the HP-UX 11i v3 and Oracle Solaris 10 environments, development tools, clustering technologies, and more. Additional sections describe some of the advanced features of Oracle Solaris that are unique in the industry and may be unfamiliar to developers new to the platform. References to more detailed information are provided throughout the document and in Appendix G.

Oracle Solaris

For over two decades, Oracle Solaris has been the platform of choice for enterprise developers and ISVs. Providing a rich environment for strategic applications, Oracle Solaris combines key computing elements—operating system, networking, storage management, and user environment—into a stable, high-quality foundation that developers can depend on for creating and deploying solutions. Many innovations, including built-in virtualization technology, support for SPARC and x86 processor-based systems, massive scalability, rich security capabilities, debugging and analysis tools and more, make Oracle Solaris the best platform for developing and deploying enterprise applications.

The Advantages of Porting to Oracle Solaris

Oracle Solaris isn't simply an operating system. It is the foundation for enterprise application development and deployment. Integrated with this powerful operating system is Oracle's software portfolio—virtualization, clustering, and development technologies such as Oracle VM, Oracle Solaris Cluster, and Oracle Solaris Studio development tools—which together form the core of a large developer ecosystem. Porting applications to this rich environment delivers a host of advantages.

- **Develop and deploy on a single platform.** Technology advancements come together in Oracle Solaris to create a single, integrated platform that developers can use to build and test applications, and model production environments. Integrated testing and patch verification across the hardware and software stack enable Oracle Solaris to offer stability for the end-to-end process of enterprise application development and deployment.
- **Scale up and out without source code modification.** Oracle Solaris runs on a broad range of SPARC and x86 processor-based systems. In addition, integrated virtualization and resource control mechanisms support the vertical and horizontal scalability and optimized utilization needed for high-demand enterprise applications.
- **Take advantage of industry-leading performance.** Oracle Solaris continues to power Oracle's x86 and SPARC servers to new world records for performance, scalability, and cost-effectiveness. An optimized TCP/IP stack, Multiple Page Size Support (MPSS), Memory Placement Optimization (MPO), multithreading advancements, and more help tune platforms and optimize systems.
- **Run with confidence.** Knowing applications can run in the face of disaster or unwelcome intrusions is essential to success. The unparalleled security features in Oracle Solaris, combined with high availability technology, deliver the reliability needed for application and service continuity.
- **Trust in the future.** In today's challenging economy, many vendors are re-evaluating product portfolios and discontinuing development. Oracle has increased investment in Oracle Solaris and plans to continue to develop innovative technologies and enhance the operating system. In addition, Oracle's unwavering source code and binary compatibility guarantee ensures that applications ported to Oracle Solaris today will run unmodified on future versions of Oracle Solaris as they emerge.

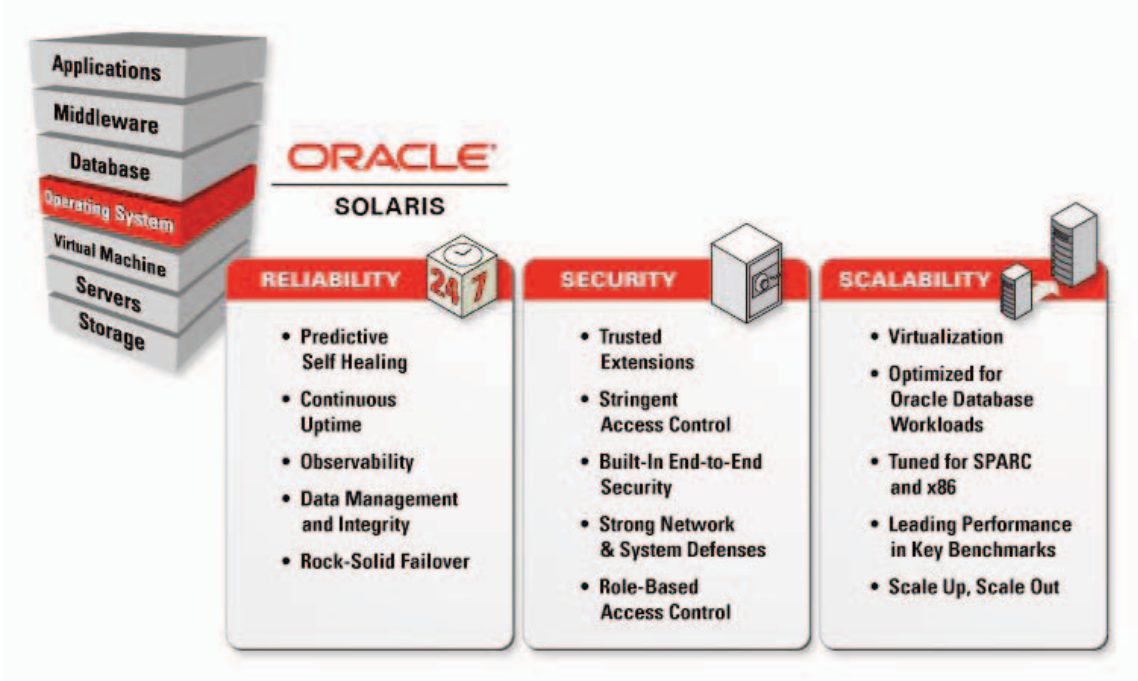


Figure 1-1. Oracle Solaris 10 delivers scalability, reliability, and security to enterprise applications.

Chapter 2 The Porting Process

While most common off-the-shelf applications used in HP-UX 11i v3 environments are available for Oracle Solaris, other applications may need to be ported when moving from HP servers based on Intel Itanium processors to Oracle's x86 or SPARC processor-based systems. Porting involves moving an application, written in one or more programming languages, to another operating system. In its simplest form, porting requires application recompilation and verification.

Most porting projects are not that simple. Differences in the development and runtime environments, application programming interface (API) divergence, and even adherence to standards can impact the porting process. In addition, reliance on specific hardware features can raise interoperability issues. In some cases, entirely different technologies and mechanisms exist to accomplish tasks or deploy applications, such as virtualization technologies, service management and clustering infrastructure, and offer new opportunities for enhancing applications and services when moving to new platforms.

Figure 2-1 identifies the key steps associated with porting applications to Oracle Solaris.

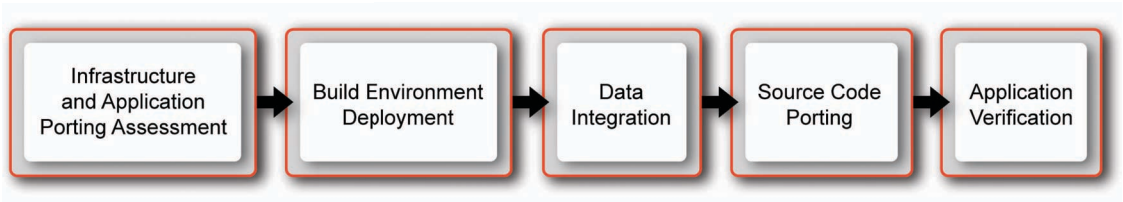


Figure 2-1. Porting an application from HP-UX to Oracle Solaris involves several steps.

Infrastructure and Application Porting Assessment

Before starting a porting effort, it is important to conduct a detailed analysis of the application and current build environment with portability in mind.

- Review all source code, build logs, shell scripts, and makefiles for the application. Creating a list of suspected porting issues and migration assumptions can prove useful during the porting process.
- Evaluate the existing and target development environments. While some developers choose to migrate open source compilers and third-party products to Oracle Solaris, most utilize platform-certified development tools to code and build applications, such as Oracle Solaris Studio. Care must be taken to understand the semantics of build options, ensuring Oracle Solaris Studio tools provide expected functionality. For example, position-independent code, static linking, extended symbol information and other techniques may require new and different options in Oracle tools.
- Understand code composition and the files used to build applications. For example, large legacy applications comprised of millions of lines of code can consist of many file types in a complicated source tree layout. Identifying and eliminating unused code in the source tree can save valuable porting time.

Build Environment Deployment

Replicating the build environment—tools, scripts, and utilities—as closely as possible can help minimize configuration and other inadvertent errors that can affect the development process. Sometimes changes must be made to account for differences between the HP-UX and Oracle Solaris environments.

- Understand the key files that affect the build system and port those files first. Creating a build log can help identify the tools, scripts, and utilities used to create an application.
- Port build scripts and make files. Try to place build tools in similar locations to minimize changes to source files, scripts, and make files.
- Modify source files to use the new tools, utilities, and libraries. Take care to ensure the options used provide the intended functionality. If the original build environment is designed well, only a few key makefile and setup scripts likely need modification.
- Perform a global search for hardcoded values in make files and change them so they benefit from the new environment. Port any remaining disconnected hardcoded instances.
- Install middleware or other third-party software needed to build the application. For example, ensure a Java™ Virtual Machine (JVM) is installed, if needed.

Data Integration

Many enterprise applications rely on information stored in databases to satisfy user requests. Because these databases contain valuable business information, it is imperative that they be migrated to a new environment with minimal disruption. Fortunately, the most popular databases, including Oracle Database and the open source MySQL database, run on HP-UX and Oracle Solaris, ensuring the replacement environment is a near one-to-one mapping of component technology.

As developers look to validate in-house enterprise applications, it may be necessary to test against large-scale data sets that mimic production systems. When doing so, be sure to:

- Acquire and install appropriate products and licenses, including the database communication layer (database client and server libraries), the embedded SQL precompiler, the C compiler and linker, and the database engine.
- Modify application source code to reflect any API changes introduced in the database technology.
- Create the database objects to accept the data, if necessary.
- Extract data from the original system and load it into the new environment, performing any needed data translation.

Source Code Porting

During the porting phase, all necessary changes to source code are implemented to ensure the application compiles and builds on Oracle Solaris. This process includes:

- Changes to source code and shell scripts
- A clean compile and build of the source code per the build environment, using the original build logs as a reference
- Checks for embedded system commands in SQL code and any needed porting
- Checks for embedded system commands or other system dependencies in scripts from other languages, such as Perl, and any needed porting
- A scan of application supporting files for system dependencies and porting

It is possible to encounter APIs in use by applications that are incompatible with Oracle Solaris. While in-line source code changes can be made, Oracle recommends creating a compatibility library that implements the changes needed to resolve any incompatibilities. Modifications to the source code can then be limited to conditional compilation directives, ensuring backward compatibility.

Application Verification

The majority of problems encountered during functional testing are related to the configuration of the runtime environment. Oracle recommends using a wide variety of techniques to ensure the widest coverage, including unit, build process, non-regression, integration, stress, and performance testing.

Commercial Applications and Third-Party Products

All applications, whether running in the HP-UX or Oracle Solaris environment, depend on support from the operating system and its associated utilities to perform work. However, some applications assume the availability of third-party products that are integrated into the HP-UX environment. When migrating to Oracle Solaris, this supporting software must be ported as well, as part of the application infrastructure. As a result, care should be taken to ensure required third-party software, or similar products, are available for Oracle Solaris. In the unlikely event that needed third-party applications are not available for Oracle Solaris, organizations can choose to make changes to the system to enable products that provide similar functionality to be used.

Chapter 3 Operating System Considerations

Developers porting applications from HP-UX 11i v3 to Oracle Solaris can take comfort in the fact that the two operating environments share a common heritage. Both are 64-bit versions of UNIX, with common APIs, languages, protocols, and data models. Because HP-UX and Oracle Solaris have common underlying architectural and design philosophies, programmers can leverage expertise across the porting process. Yet the porting of applications is never a trivial issue, even between similar systems such as HP-UX and Oracle Solaris. As a result, developers should consider how applications and development processes are affected by potential differences in several areas.

Processor Endianness

Processors store multibyte data objects, such as an 8-byte integer, in one of two ways. Architectures that store the most-significant byte of the data first are referred to as Big Endian (BE), while those that store the least-significant byte first are called Little Endian (LE). HP-UX 11i uses the Big Endian convention. Oracle Solaris supports both modes. For example, Oracle Solaris uses a Big Endian architecture on SPARC processor-based systems and a Little Endian architecture on x86 platforms.

When standard UNIX APIs are used to access data, the endianness of the platform usually is not an issue as data layout is abstracted by the APIs used to access it. Use of lower-level languages, legacy coding techniques, or operators such as *shifts*, logical or arithmetic *ands* and *ors*, with assumptions about the layout of a data element (and the consequential significance of the bits) can be an issue. The use of overlaid data structures and casts may require an understanding of the logic of the application, and possible code modification.

The endianness of the processor and operating system do not directly affect application performance or scalability, and the migration from HP-UX to Oracle Solaris should not pose undue migration risk due to operating system endianness. Lower level code, such as a device driver or other control software, may be sensitive to endianness. When faced with such challenges, developers can draw on many resources, such as sample device drivers, the Device Driver Interface and Driver Kernel Interface (DDI/DKI), derived types, and the help of Oracle experts to assist in the porting effort.

Data Alignment

The processor that runs in a computing system has a natural, or preferred, primary memory alignment for data objects. Oracle Solaris makes use of this fact when dealing with data alignment. For example, on Oracle systems with 64-bit SPARC T3 processors, Oracle Solaris loads and stores a 64-bit (8-byte) pointer on 8-byte primary memory boundaries in order to take advantage of the underlying hardware. With a compound data type such as a C structure, which may contain a mix of differing data sizes, it is sometimes necessary to insert bytes for extra padding between data elements to satisfy processor data alignment rules. The use of padding can be specified with a compiler option, enabling padding to be permitted or disallowed as needed for space or performance optimization.

When porting an application from HP-UX 11i to Oracle Solaris, the differences in natural data alignment of basic data types typically is not an issue as the compiler and APIs handle data type alignment correctly. The use of overlaid data structures and casts (or equivalent) require an understanding of the logic of the application, and possible code modification.

Read/Write Structures

Most applications read and write data to permanent storage media as a complete structure in binary form using standard C or C++ routines. These routines need the data size that is being read or written. Due to the different alignment and data type sizes on PA-RISC, Intel Itanium, SPARC, and x86 platforms, structure sizes vary. To ensure portability, the `sizeof()` operator should be used to specify the number of bytes to read or write in these routines.

Storage Order and Alignment

The order of data storage varies between platforms, and source code that expects a particular storage order is not portable. To maximize portability, source code should compare the individual fields of structure variables separately, rather than relying upon storage order.

Because data alignment and datatype sizes vary based on the platform, the alignment of structure field members also differs across platforms. This results in variable padding requirements that cause structures to change size. To maximize portability, the structure size should be accessed using the `sizeof()` operator.

64-Bit Data Models

Organizations are reaping the performance advantages of keeping more application data in main memory. Although few require an entire 64-bit address space, the impact of more than 32 bits of address space benefits a wide variety of commercial and high-performance computing applications. For example, databases, Web caches, simulation, and modeling software run more effectively in the much larger primary address space of a 64-bit architecture. These benefits include:

- A greater proportion of a database can live in primary memory
- Larger CAD/CAE models and simulations can live in primary memory
- Larger scientific computing problems can fit in primary memory
- Web caches can hold more data in primary memory and reduce access latency
- The wider data paths of a 64-bit processor offer improved computational performance

HP-UX and Oracle Solaris support 64-bit computing using the LP64 data model, in which longs and pointers are 64 bits in length. With the same 64-bit data model, application modifications are minimized when migrating from HP-UX to Oracle Solaris. Both operating system kernels provide the larger data

paths and primary address space that 64-bit applications demand. As a result, applications that rely on these features are well positioned to take advantage of the scalability, manageability, and reliability provided by Oracle systems.

Not all applications are well suited to a 64-bit model. If the application does not require a large primary address space or wider data path, migrating the application to the 64-bit model may not be warranted. In such situations, Oracle Solaris provides support for applications written using a 32-bit computing model, enabling a smooth transition yet providing access to 64-bit computing and other features should the need arise.

Best Practices for Converting C and C++ Applications to the LP64 Data Model

It is important to note that many size issues can be mitigated when migrating between data models by using derived types, a mechanism for specifying the size of an attribute or its intended use. The use of derived types — which are safe for both 32- and 64-bit environments — increases program clarity and portability, helping to mitigate differences in the hardware environment. In addition, consider the following best practices when porting 32-bit C applications to the LP64 data model.

- **Integer and pointer size change.** Some 32-bit code relies on integers and pointers being the same size. Pointers are often cast to `int` or `unsigned int` for address arithmetic. Instead, cast pointers to `long`. Long and pointers are the same size in the ILP32 and LP64 data models. Rather than explicitly using `unsigned long`, use `uintptr_t`, as it expresses the intent more closely and makes code more portable, insulating it against future changes.
- **Integer and long size change.** Because integers and longs are identical in a 32-bit model, developers often use them interchangeably. When porting, ensure the use of integers and longs conforms to ILP32 and LP64 data model requirements. While an integer and a long are both 32-bits in the ILP32 model, a long is 64 bits in the LP64 model.
- **Arrays.** Sometimes developers use large arrays of longs or unsigned longs rather than arrays of integers or unsigned integers. Using these wider arrays can degrade performance in the LP64 model. If arrays of integers work in well the application, use them instead of arrays of longs or arrays of pointers.
- **Sign extension.** Conversion to 64-bit compilation often causes sign extension due to rules governing type conversion and promotion. Use explicit casting to prevent sign extension issues and achieve intended results.
- **Pointer arithmetic.** In general, pointer arithmetic works better than integer arithmetic as it is independent of the data model.
- **Structures.** Check the application's internal data structures for holes. Insert padding between structure fields to meet alignment requirements. Padding allocation occurs as long or pointer fields expand to 64 bits in the LP64 data model. In the 64-bit compilation environment on SPARC platforms, structures are aligned to the size of their largest member. When repacking a structure, move long and pointer fields to the beginning of the structure.
- **Unions.** Check unions, as their fields can change size between the 32-bit and 64-bit models.

- **Type constants.** Data loss can occur in some constant expressions due to a lack of precision. Explicitly specify the data types in a constant expression. Specify the constant type or use casts to specify the type of a constant expression.
- **`sizeof()` return value.** In the LP64 data model, `sizeof()` has the effective type of an unsigned long. Occasionally, `sizeof()` is passed to a function expecting an argument of type `int` or something assigned or cast to an integer. This truncation can cause data loss.
- **Format string conversion operation.** Ensure the `printf(3S)`, `sprintf(3S)`, `scanf(3S)`, and `sscanf(3S)` format strings accommodate long or pointer arguments. For pointer arguments, specify `%p` for the conversion operation in the format string to ensure the code works in 32-bit and 64-bit compilation environments.

Additional information regarding migration from 32-bit to 64-bit environments can be found in "Converting 32-bit Applications Into 64-bit Applications: Things to Consider" at http://www.oracle.com/technetwork/server-storage/solaris/ilp32tolp64issues-137107.html.

Chapter 4 Runtime Environment

Environment Variables

HP-UX 11i v3 and Oracle Solaris use environment variables to convey information to applications, such paths to libraries. Table 4-1 lists the key environment variable of interest to application developers.

TABLE 4-1. KEY ENVIRONMENT VARIABLES

DESCRIPTION	HP-UX 11i v3	ORACLE SOLARIS 10	NOTES
Library path	`LD_LIBRARY_PATH`	`LD_LIBRARY_PATH`	• Enables use of a different library without relinking application • Checked by default in both operating systems • Use not recommended for production code
Search path	`PATH`	`PATH`	• Identifies the paths to search, in search order
Compiler options	`CFLAGS`	`CFLAGS`	• Lists compiler options to use
Compiler	`CC`	`CC`	• Identifies the compiler to use
Linker options	`LDFLAGS`	`LDFLAGS`	• Lists linker options to use
Home directory	`HOME`	`HOME`	• Identifies the user's home directory
Shell	`SHELL`	`SHELL`	• Identifies the user's preferred shell

Permissions

Most enterprise applications create files in the course of execution. Default directory permissions and file mode creation mask (umask) settings can impact file creation. While HP-UX 11i v3 uses a default umask setting of 000, Oracle Solaris 10 sets a default umask of 022 in the `/etc/profile` file. Be sure to check key application files, including configuration and logfiles, to ensure they have the proper settings. When porting applications, use the `umask()` function to check and set the file mode creation mask to desired values.

Process Resource and Runtime Limits

HP-UX 11i and Oracle Solaris allow developers to place limits on the consumption of resources by application processes. Each limit consists of two values: a *soft* (current) limit and a *hard* (maximum) limit. Soft limits can be changed by a process, but must remain less than or equal to the hard limit. Hard limits can be lowered to a value that is greater than or equal to the soft limit. However, only processes with certain privileges can raise a hard limit. As a result, the lowering of a hard limit should be done with care.

While the process limit concepts are similar in the two environments, the implementations allow different levels of control and specify different default values in some cases. For example, Oracle Solaris 10 defines variables for controlling the soft and hard limits for the stack segment size, while HP-UX 11i v3 only defines variables for hard limits. Table 4-2 lists the key process limits and their default values that typically are of concern to application developers. The `sysdef(1M)` and `ulimit(1)` commands can be used on Oracle Solaris to obtain the current system definition and set or get limitations on the system resources available to the current shell, respectively.

TABLE 4-2. KEY PROCESS LIMIT DEFAULT VALUES

NAME	HP-UX 11i v3		ORACLE SOLARIS 10
	32-BIT	64-BIT	
`coredumpsize`	Unlimited	Unlimited	Unlimited
`cputime`	Unlimited	Unlimited	Unlimited
`datasize`	256 GB	1 MB	Unlimited
`descriptors`	1024	1024	1024
`memoryuse`	Unlimited	Unlimited	Unlimited
`stacksize`	8 MB	256 MB	Unlimited

HP-UX 11i v3 and Oracle Solaris specify runtime limits for applications. These limits are set system wide and not for a given process. Table 4-3 lists the key limits that can affect an application at runtime.

TABLE 4-3. RUNTIME LIMIT DEFAULT VALUES

DESCRIPTION	HP-UX 11I v3		ORACLE SOLARIS 10	
Soft file limit per process	`maxfiles`	Default: 60		
Hard file limit per process	`maxfiles_lim`	Default: 1024		
Maximum number of threads/process	`max_thread_proc`	Default: 256		
Maximum number of user processes	`maxuprc`	Default: 256	`maxuprc` Default: `max_nprocs` – `reserved_procs`	
Maximum number of users	Obsolete on HP-UX 11i as of v2		`maxusers` Default: 2048 or the amount of memory available, in MB (whichever is less)	
Maximum open files on the system	`nfile`	Default: 8192	`nfile`	Default:
Maximum file locks on the system	`nflocks`	Default: 200		
Maximum processes on the system			`max_nprocs`	Default: 10 + (16 x `maxusers`)

TABLE 4-3. RUNTIME LIMIT DEFAULT VALUES

System process slots to reserve in the process table for root processes	`reserved_procs`	Default: 5

Application Programming Interfaces

HP-UX 11i v3 and Oracle Solaris follow the POSIX standards and provide a well-defined system call interface to access kernel facilities. While most of the calls are identical on these systems, there are some differences. In some cases, functions with the same name take different parameters, or behave differently, depending on the platform. Also, some calls are available only on HP-UX 11i v3, or only on Oracle Solaris. Appendix B lists some of the key API differences between the platforms.

Most interface functions return `-1` to indicate an error and set the global variable `errno` to provide a complete description of the cause.

System Libraries

HP-UX and Oracle Solaris provide a wide range of system libraries. While many of the libraries have the same name and provide identical functionality, there are some differences. In some cases, the libraries provide the same functionality but are named differently. If other cases, the functionality provided in a library on one system spans multiple libraries on the other system. Appendix A provides a list of key libraries and their equivalence on the HP-UX 11i v3 and Oracle Solaris platforms.

Shells and Utilities

Oracle Solaris provides a number of shells and utilities for developers, including:

- **Bourne shell.** The Bourne shell (`/usr/bin/sh`) is the default shell in Oracle Solaris 10. The same path is used on HP-UX 11i v3 to link to the POSIX shell, a superset of the Bourne shell. Several POSIX shell options are not supported in the Bourne shell, and some commands behave differently. For example, the Bourne shell does not support the `–p` option to the `export` command, and the output of the `hash` command is different in the two shells. More information on the Bourne shell can be found in the `sh(1)` man page.
- **C shell.** HP-UX 11i v3 and Oracle Solaris support the C shell (`/usr/bin/csh`). Most commands and options are identical on the two platforms. More information on the C shell can be found in the `csh(1)` man page. On Oracle Solaris, it is considered a best practice to avoid writing scripts in `csh`.
- **Korn shell.** HP-UX 11i v3 and Oracle Solaris support the Korn shell *(*`ksh88` located in `/usr/bin/ksh`). While the implementation is largely the same in both environments, there are a few differences. For example, larger array subscripting is supported on Oracle Solaris, and some commands produce slightly different output. More information on the Korn shell can be found in the `ksh(1)` man page. (Note that Oracle Solaris 11 Express uses `ksh93`.)

- **`appcert` utility.** The `appcert(1)` utility helps identify potential binary compatibility issues when porting applications to Oracle Solaris by examining an application's conformance to the Oracle Solaris Application Binary Interface (ABI).

Scripts

The UNIX environment has a long history of multiple tools or utilities working together to accomplish a larger task. Developers often use scripts to automate compilation, create application administration tools, analyze or modify data, and provide functional support for an application. Note that scripts do not just do the work themselves—they often leverage programs and utilities that exist elsewhere in the operating environment to perform various administrative tasks. While there are great similarities between the HP-UX and Oracle Solaris scripting environments, it is important to be aware of possible differences. Some possible issues include:

- Command not available
- Command is in a different location
- Command uses an option, or flag, which does not exist in Oracle Solaris
- Command uses an option, or flag, which provides different functionality in Oracle Solaris
- Command output is different and/or redirected to a different location

Appendix F provides a summary of key command differences on HP-UX 11i v3 and Oracle Solaris.

Chapter 5 Devices

In the event an application makes use of devices that are not natively supported on by the operating system, a device driver or software module likely needs to be ported. HP-UX and Oracle Solaris provide a similar device driver model and architecture, easing porting issues when such specialized, custom software is needed. In Oracle Solaris, device drivers are loadable modules that are dynamically loaded into memory by the kernel when they are needed.

The kernel provides access to device drivers through the following features:

- **Device-to-driver mapping.** The kernel maintains the device tree. Each node in the tree represents a virtual or a physical device. The kernel binds each node to a driver by matching the device node name with the set of drivers installed in the system. The device is made accessible to applications only if a driver binding exists.
- **Device Driver Interface/Driver-Kernel Interface (DDI/DKI).** The DDI/DKI interfaces standardize interactions between a device driver and the kernel, the device hardware, and the boot and configuration software. These interfaces keep the device driver independent from the kernel and improve its portability across successive releases of the operating system on a particular machine.
- **Layered Driver Interface (LDI).** The LDI is an extension of the DDI/DKI that enables a kernel module to access other devices in the system. The LDI also provides a mechanism for determining which devices are in use by the kernel.

Device Naming Conventions

HP-UX 11i v3 and Oracle Solaris use device names as a means to give users and applications controlled access to devices and critical system resources such as disk drives, network adaptors, memory and I/O channels. Both operating systems place device resources into a highly secure file name hierarchy used for files and directories, bringing the full power of the security system to device access and control. By treating device access just like file access, existing, proven Oracle Solaris security models are automatically applied, closing potentially devastating back-door security flaws that can render data vulnerable to unauthorized users or applications.

While the actual directory placement and naming of devices varies when migrating from HP-UX 11i v3 to Oracle Solaris, the device security and control semantics—involving operating system calls such as `open()`, `ioctl()`, and `close()`—are the same. The result is that porting an application that interacts with device control software, such a device driver, from HP-UX 11i to Oracle Solaris is straightforward. Developers only need to change the file system hierarchy location and device name in the source code and recompile on Oracle Solaris. Program control logic need not be modified.

Device Paths

Table 5-1 lists the disk and tape device naming conventions for HP-UX 11i v3 and Oracle Solaris.

TABLE 5-1. DEVICE PATHS

DESCRIPTION	HP-UX 11i v3	ORACLE SOLARIS 10
Disk (Block Access)	`/dev/disk/disk#`	`/dev/dsk/c#t#d#p#`
Disk (Raw Access)	`/dev/rdisk/disk#`	`/dev/rdsk/c#t#d#p#`
Disk Partition (Block Access)	`/dev/disk/disk#_p#`	`/dev/dsk/c#t#d#p#`
Disk Partition (Raw Access)	`/dev/rdisk/disk#_p#`	`/dev/rdsk/c#t#d#p#`
Tape (Raw Access)	`/dev/rtape/tape#options`	`/dev/rmt/#`

Device Driver Interface/Driver Kernel Interface

The Oracle Solaris DDI/DKI enables the development of device drivers that operate with full source-level compatibility across multiple platforms and instruction sets architectures, including x86 and SPARC. Host bus dependencies must be removed from the device driver in order to produce multiple platform, multiple instruction set architecture portability.

The mature DDI/DKI provides operating system resources, configuration properties, device mapping, and low-level I/O access to device drivers. Use of the DDI/DKI provides an abstraction layer, helping to isolate the device driver from hardware, as well as offering portability, cleaner control logic, and a consistent interface across similar devices. Device drivers maintain their portability by adhering to a standard suite of interfaces rather than directly accessing the system resource or I/O devices, and the issue of working with 32- and 64-bit data is greatly simplified.

Platform independence is accomplished by the design of the DDI/DKI in the following areas:

- Dynamic loading and unloading of modules
- Power management
- Interrupt handling
- Accessing the device space from the kernel or a user process (register and memory mapping)
- Accessing kernel or user process space from the device using DMA services
- Managing device properties

Oracle provides sample device drivers, documentation, installation instructions, operating instructions and downloadable tools that support device driver development efforts for Oracle Solaris platforms. See http://download.oracle.com/docs/cd/E19253-01/index.html for more information.

Best Practices for Porting Device Drivers

The porting effort for device drivers requires extra care, as drivers interact directly with hardware and operate without the protection of the operating system afforded to user processes. By building debugging support into the device driver during the porting effort, developers can simplify porting, maintenance work, and future development.

- **Use a unique prefix to avoid kernel symbol collisions.** Since each driver module is linked into the kernel, symbol names for drivers must not collide with other kernel symbols. To avoid collisions, add a driver-specific prefix, such as the name of the driver, to each function and data element.
- **Use the `cmm_err()` function to log driver activity.** Use this function to print messages to a system log from the device driver to facilitate debugging. Messages are placed in the `/var/adm/messages` file.
- **Use the `ASSERT()` macro to catch invalid assumptions.** Assumptions are a valuable form of active documentation. The `ASSERT()` macro halts kernel execution if a condition that is expected to be true is false, providing a mechanism for validating assumption made in source code.
- **Use `mutex_owned()` to validate and document locking requirements.** A significant portion of device driver development involves properly handling multiple threads. The `mutex_owned()` function can be used to determine if a mutex is held by a thread. Note this function is valid only within `ASSERT()` macros.
- **Use conditional compilation to toggle expensive debugging features.** Debugging code can be placed in a device driver by conditionally compiling code based on a preprocessor symbol, such as `DEBUG`, or by using a global variable. Conditional compilation offers an advantage: unnecessary code can be removed in the production driver. Using a global variable allows the amount of debugging output to be chosen at runtime. This can be accomplished by setting a debugging level at runtime with an `ioctl` or through a debugger. Typically these two methods are combined.

Use Defensive Programming

Several defensive programming techniques can be used to prevent device driver source code from causing system panics or hangs, draining system resources, or allowing the spread of corrupt data. It is recommended that Oracle Solaris device drivers adhere to the following coding practices.

- **Control only one piece of hardware.** Each piece of hardware should be controlled by a separate instance of a device driver. While an instance has its own data space, it shares text and global data with other instances. Drivers should use a separate instance for each piece of hardware unless the driver is designed to handle failover internally.
- **Take care with programmed I/O.** Perform programmed I/O (PIO) through the device driver interface (DDI) access functions, using the appropriate data access handle.
- **Assume data received from a device could be corrupt.** Device drivers should check the integrity of data before using it. In particular, extreme care should be taken with pointers, memory offsets, or array

indexes read or calculated from data supplied by the device. Such values can cause a kernel panic if dereferenced, and should be checked for range and alignment (if required) before use.

- **Avoid releasing bad data to the rest of the system.** Device errors can result in corrupt data being placed in receive buffers. Such corruption is indistinguishable from corruption that occurs beyond the domain of the device, for example within a network. Typically, existing software is in place to look for corruption, such as integrity checks at the transport layer of a protocol stack or within the application using the device. If data integrity checks are not going to be performed at a higher software layer, they should be performed within the device driver using device-specific techniques, such as validating checksums, cyclical redundancy checks (CRCs), and so on.
- **Use only documented DDI functions and interfaces.** Documented functions and interfaces provide greater stability and ensure code portability.
- **Keep memory pages clean.** Ensure all writes by the device into DMA buffers are contained within pages of memory controlled entirely by the driver. This prevents a DMA fault from corrupting an arbitrary part of the system's main memory.
- **Be respectful of system resources.** The device driver must not be an unlimited drain on system resources should the device hang. It should time out if a device claims to be continuously busy. The driver should also detect a pathological (stuck) interrupt request and take appropriate action.
- **Support Oracle Solaris hot-plug capabilities.** Support for hot-plugging is essential in enterprise environments where components often are moved to affect better resource utilization or to replace a failed component.
- **Free resources after a fault.** The operating system must be able to close all minor devices and detach driver instances even after hardware fails.

More information can be found in the *Writing Device Drivers* manual in the Oracle Solaris documentation set located at http://download.oracle.com/docs/cd/E19253-01/816-4854/index.html.

Chapter 6 Development Environment

Porting, verifying, and optimizing applications can be a time-consuming and complex process. While individual point products can help with certain tasks, building applications with an integrated platform in which all of the pieces work together streamlines workflow and results in more robust applications. Oracle Solaris Studio 12.2 provides everything needed to create high quality, cross-platform desktop, enterprise, and Web applications. An integrated development environment optimizes the application development process, from creating and building C, C++, Java, or Fortran applications, to debugging problems and tuning for optimal performance. By integrating all the steps programmers take—from GUI design and code generation, to edit-compile-debug-tune cycles—the Oracle Solaris Studio integrated development environment (IDE) makes it easy to rapidly port enterprise applications.

Oracle Solaris Studio is designed to:

- Maximize application performance with optimizing compilers
- Simplify multicore development with automatic parallelization features and advanced tools
- Improve productivity with a next-generation IDE and tools with rich graphical interfaces
- Simplify development across multiple architectures (SPARC and x86) and operating systems (Oracle Solaris and Linux)

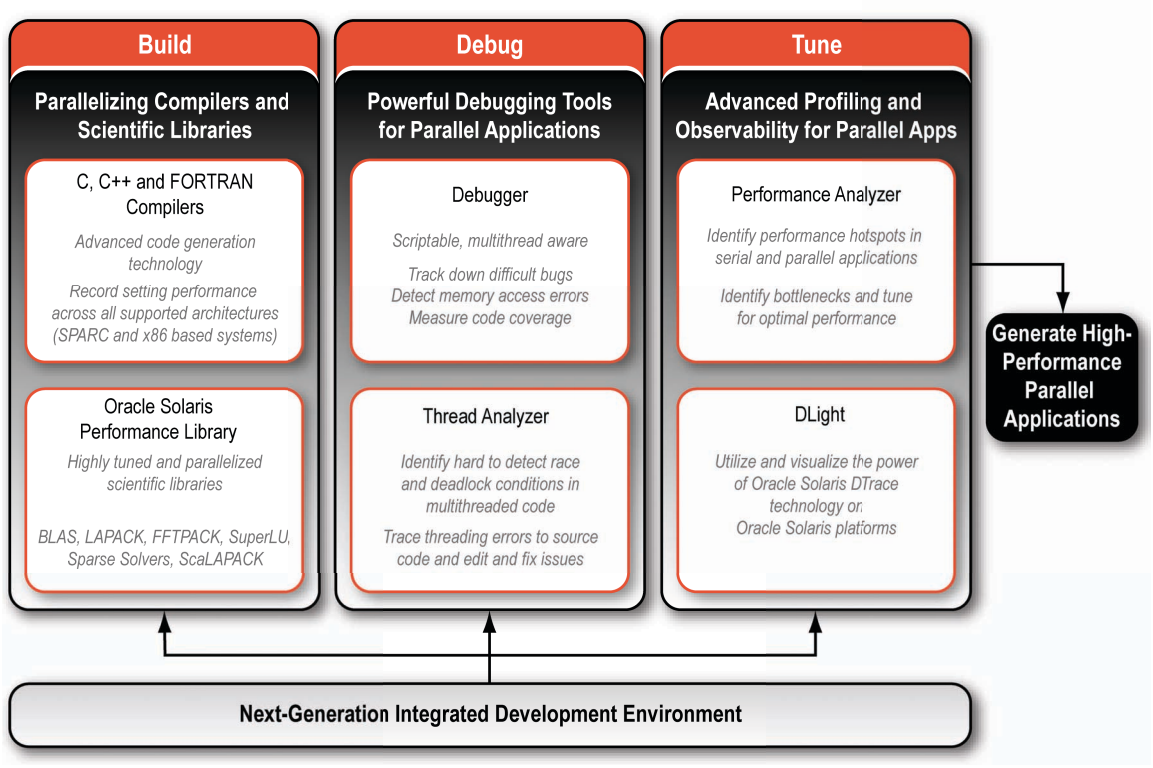


Figure 6-1. The next-generation Oracle Solaris Studio IDE integrates advanced tools for application development.

Oracle Solaris Studio Components

Oracle Solaris Studio offers a comprehensive set of development tools.

- **Optimizing C, C++, and Fortran compilers.** The Oracle Solaris Studio compilers generate improved application performance on Intel x86, AMD x86, UltraSPARC, and SPARC64 processor-based systems. With a wealth of recent industry-based benchmarks, Oracle Solaris Studio compilers take full advantage of the latest multicore architectures.
- **Full OpenMP 3.0 compiler, debugger, and tools support.** The OpenMP 3.0 specification contains new features to ease multicore development, and takes a more general approach to multithreaded programming by using tasks to support complex and dynamic control flows.
- **Sun Memory Error Discovery Tool (Discover).** Memory-related errors in programs can be the most difficult for developers to debug and frequently are the cause of erratic program behavior. The Sun Memory Error Discovery Tool is used to instrument a binary, enabling errors to be caught and reported dynamically as the program executes.
- **Code Coverage Tool (Uncover).** Integrated into the Performance Analyzer, this is simple and easy-to-use tool for measuring code coverage makes it possible to quickly find major functional areas within binaries that are not being tested.
- **DLight.** System profiling tools allow developers to explore systems, understand how they work, and identify performance problems across many software layers. DLight is a new tool that unifies application profiling and system profiling using Oracle Solaris DTrace technology on Oracle Solaris platforms.
- **dbxTool.** The dbx debugger is fully integrated into the IDE and is available via the command line. Oracle Solaris Studio 12.x features dbxtool, a stand-alone debugging solution with a user-friendly interface. With dbxtool, developers can quickly and easily debug an executable or core file, or attach to a running process.
- **Thread Analyzer.** The Thread Analyzer is a tool that helps identify common—yet notoriously difficult to debug—issues in multithreaded code. It analyzes program execution across multiple threads and detects data race and deadlock conditions.
- **Performance Analyzer support for MPI applications.** The Oracle Solaris Studio Performance Analyzer includes an MPI Timeline and MPI charts, along with zooming and filtering capabilities.
- **Updated Oracle Solaris Studio IDE.** Oracle Solaris Studio features a next-generation IDE based on NetBeans 6.5.1 software, specifically geared for C/C++ developers. New features include improved code completion, error highlighting, semantic highlighting, call graph, memory window, packaging of application as tar files, zip files, System V Release 4 (SVR4 packages), RPMs, or Debian packages, and much more.
- **Sun Performance Library.** The Sun Performance Library is a set of optimized, high-speed mathematical subroutines for solving linear algebra and other numerically-intensive problems. Developers can use these routines for solving computational linear algebra, fast Fourier transforms

(FFTs), and other numerically intensive problems. Oracle Solaris Performance Library routines are callable from Fortran, C, and C++ programs, and contain Oracle's enhanced implementations of the routines in BLAS1, BLAS2, BLAS3, LAPACK, ScaLAPACK, PBLAS, BLACS, FFTPACK, VFFTPACK, SPSOLVE, Sparse BLAS, and SuperLU.

Oracle Solaris Studio Workflow

Figure 6-2 illustrates a typical Oracle Solaris Studio IDE workflow versus a standalone tool process.

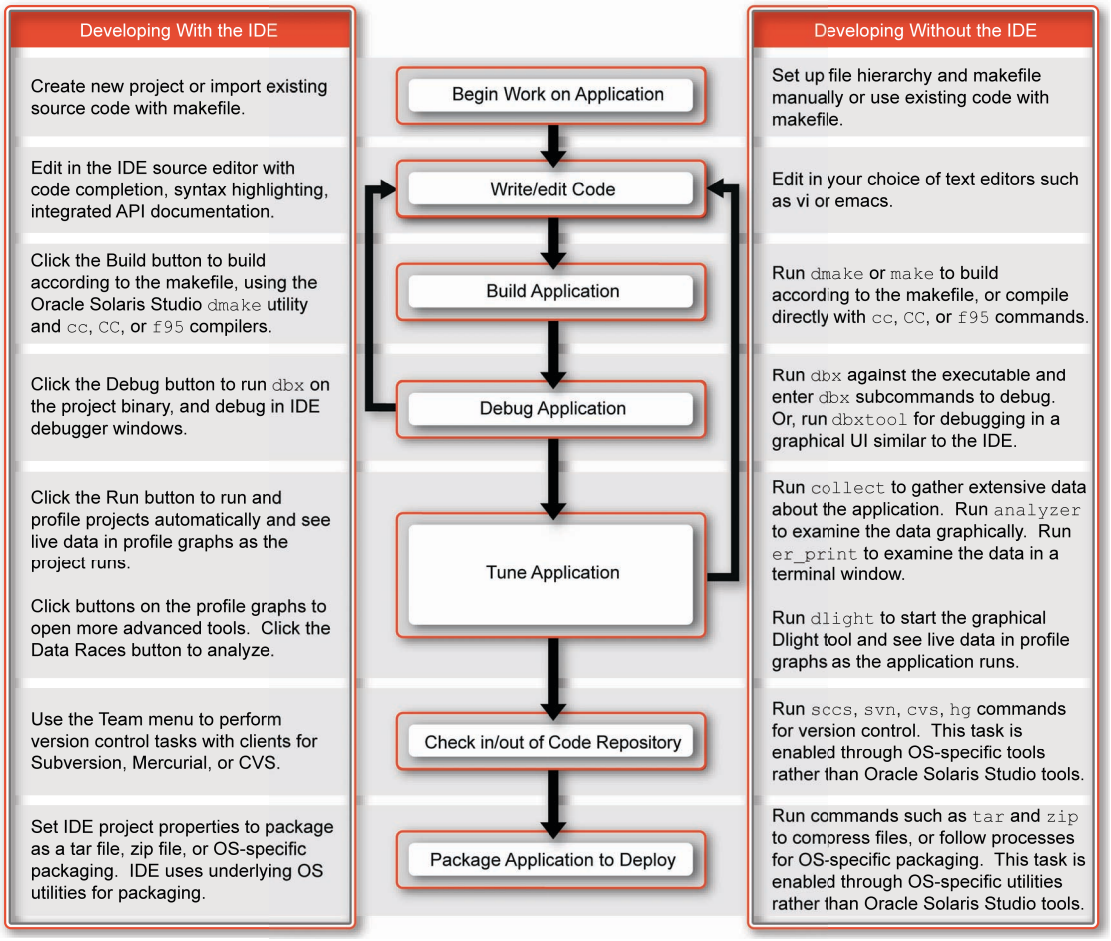


Figure 6-2. The Oracle Solaris Studio IDE workflow optimizes the development and tuning process.

Supported Platforms

Oracle Solaris Studio supports systems that use the SPARC and x86 families of processor architectures: UltraSPARC, SPARC64, AMD64, Intel® Pentium®, and Intel® 64.

Supported Standards

Changes to source code can be required as a result of the semantic and syntactic differences in how Oracle and HP implement the C, C++ and Fortran languages.

- **C compiler.** The Oracle Solaris Studio C compiler conforms to the ISO/IEC 9899:1999 (*Programming Languages—C*) and ISO/IEC 9899:1990 (*Programming Languages—C*) standards. The C compiler also supports the latest OpenMP 3.0 shared-memory parallelism API.
- **C++ compiler.** The Oracle Solaris Studio C++ compiler (CC) supports the ISO International Standard for C++, ISO IS 14882:2003, *Programming Languages—C++*.
- **Fortran compiler.** The Oracle Solaris Studio Fortran compiler is a Fortran 95 compiler that conforms to published Fortran language standards, and provides many extended features, including multiprocessor parallelization, sophisticated optimized code compilation, and mixed C/Fortran language support. While some HP-UX enterprise applications were created using Fortran 95 semantics and compilers, many legacy applications used FORTRAN 77. While Oracle's Fortran 95 compiler accepts many FORTRAN 77 features directly, some may require compiling in FORTRAN 77 compatibility mode. To ensure continued portability, applications using non-standard FORTRAN 77 features should be ported to standards-conforming Fortran 95.
- **Hardware optimization.** On SPARC platforms, Oracle Solaris Studio compilers provide support for the optimization-exploiting features of the SPARC V9 architecture, including the UltraSPARC implementation. These features are defined in the *SPARC Architecture Manuals*, Version 8 (ISBN 0-13-825001-4), and Version 9 (ISBN 0-13-099227-5), published by Prentice-Hall for SPARC International.

Header Files and System Libraries

The Oracle Solaris Studio C and C++ compilers search for include files in the current working directory, any directories specified during compilation (`–I` option), and the `/usr/include` directory.

The Oracle Solaris 10 operating system installs several libraries in the `/usr/lib` directory. Most of these libraries have a C interface. Of these, the `libc` and `libm`, libraries are linked to applications by default. In addition, the C++ compiler includes several runtime support libraries. Some of these libraries are available only in compatibility mode (`-compat=4`), some are available only in the standard mode (`-compat=5`), and some are available in both modes (Table 6-1).

TABLE 6-1. ADDITIONAL C++ LIBRARIES.

LIBRARY	DESCRIPTION	COMPATIBILITY
`libstlport`	STLport implemenation of the standard library	`-compat=5`
`libstlport_dbg`	STLport library for debug mode	`-compat=5`
`libCrun`	C++ runtime	`-compat=5`

`libCstd`	C++ standard library	`-compat=5`
`libiostream`	Classic iostreams	`-compat=5`
`libC`	C++ runtime, classic iostreams	`-compat=4`
`libcsunimath`	Supports the `–xia` option	`-compat=5`
`libcomplex`	Complex number library	`-compat=4`
`librwtool`	Tools.h++ 7	`-compat=4, -compat=5`
`librwtool_dbg`	Debug-enabled Tools.h++ 7	`-compat=4, -compat=5`
`libgc`	Garbage collection	C interface
`libdemangle`	Demangling	C interface

Java Programming Tools

Oracle provides a comprehensive set of tools for developers creating enterprise-class Java applications.

- **Java Platform, Standard Edition (Java SE).** Java SE is a toolkit for developing all Java applications not intended for consumer devices. Java SE includes a compiler, runtime environment, and core API.
- **Java Platform, Enterprise Edition (Java EE).** Java EE builds on Java SE and adds an application server, Web server, Java 2 Platform, Enterprise Edition API, support for Enterprise JavaBeans, Java servlets API, and JavaServer Pages (JSP) technology.
- **JavaFX.** The JavaFX platform is the evolution of the Java client platform designed to enable application developers to easily create and deploy rich Internet applications that behave consistently across multiple platforms. Built on Java technology, the JavaFX platform provides a rich set of graphics and media APIs with high-performance hardware-accelerated graphics and media engines that simplify development of data-driven enterprise client applications. More information on the JavaFX platform can be found at http://download.oracle.com/javafx/2.0/overview/jfxpub-overview.htm.
- **NetBeans IDE.** A free, open-source environment, the NetBeans IDE supports the creation of enterprise, Web, desktop, and mobile Java applications. All IDE tools and features are fully integrated to improve developer productivity.
- **Oracle JDeveloper.** Oracle JDeveloper is a free, integrated development environment that simplifies the development of Java-based SOA applications and user interfaces with support for the full development life cycle.

More information on Oracle's extensive portfolio of Java technology offerings and tools can be found at http://www.oracle.com/java.

Developing Applications

HP and Oracle offer C and C++ compilers that offer similar functionality through slightly differing flags and options. By default, Oracle Solaris Studio compilers are installed in the `/opt/SUNWspro/bin` directory, while HP-UX compilers are installed in the `/opt/aCC/bin` directory. The compilers can generate 32-bit or 64-bit ELF objects, and default to 32-bit ELF objects, on both platforms.

The sections that follow discuss the considerations for porting applications from HP-UX to Oracle Solaris using the vendor-supplied C and C++ compilers. Note that the GNU compilers for C and C++ are available for both platforms, and projects committed to the GNU compilers typically port easily. However, performance-sensitive enterprise applications tend to benefit (on both platforms) from using vendor-supplied compilers.

High-Level Option Usage Strategy

Several strategies can assist the porting of high-performance, enterprise applications to Oracle Solaris Studio compilers.

- **Use reporting options and tools.** Using tools, such as `lint`, and taking advantage of the highest reporting levels can help speed the identification code issues that need attention.
- **Take advantage of precompiled header files.** Available for Oracle Solaris Studio C and C++ compilers, pre-compiled headers can improve productivity and ease the adjustment to a new project environment. (See the `–xpch` option for more information.)
- **Use debugging options to full effect.** Enabling the maximum debugging options and executing test runs can shed light on errors.
- **Optimize for performance and use an iterative process.** Set the `–fast` and other appropriate performance analysis data collection options, test, and iterate. For many enterprise applications, profile feedback, inter-procedural analysis, and other advanced options could be worth pursuing.

HP-UX and Oracle Solaris Studio C and C++ Compiler Options Comparison

In most cases, the HP-UX and Oracle Solaris Studio C and C++ compilers provide similar functionality. Table 6-2 lists available HP-UX compiler options and their Oracle Solaris Studio counterparts. Options are marked as "`N/A`" where the HP-UX compiler offers no equivalent functionality. Because the C and C++ compilers use many of the same options, Table 6-2 combines option information for both compilers. Options specific to the C or C++ compilers are noted. Additional detail on x86 architecture-specific options can be found in the *C User's Guide* for Oracle Solaris Studio.

TABLE 6-2. C AND C++ COMPILER OPTION COMPARISON

HP-UX	ORACLE SOLARIS STUDIO	DESCRIPTION
CODE GENERATION		
N/A	`-xmemalign=ab`	Controls memory alignment, a:1..8 byte alignment is possible. Generally 8 is best for performance. b: i,s or f. The default for current SPARC processors is 8i (SPARC only).
`-fast`	`-fast`[1]	Selects a good combination of compilation options for speed. Several Oracle Solaris Studio compiler options are set when using the –fast option: `-xtarget=native`, `-x05`, `-xlibmil`, `-xlibmopt`, `-xmemalign=8s (SPARC)`, `-nofstore`, `-fsimple=2`, `-fns`, `-ftrap=%none`, …
N/A	`-fma[={none|fused}]`	Enables fused multiply add for SPARC processors that support it.
`+FPVZO`	`-fnonstd`	Expands to `-fns -trap=common`.
N/A	`-fns`	Turns on SPARC nonstandard floating-point mode, allowing underflow to zero rather than gradual underflow.
See `fesetround(3M)`	`-fround=`*r*	Sets the floating-point rounding mode.
N/A	`-fsimple[=`*n*`]`	Allows the optimizer to make simplifying assumptions concerning floating-point arithmetic.
N/A	`-fstore`	Causes the compiler to convert the value of a floating-point expression or function (x86 only).

`+FPstring`
`+FPVZO`
	`-ftrap=`*t*
	`-ftrap=common`
	`-fns`
		Sets the IEEE 754 trapping mode.

`+Z/+z`	`-xcode=pic32/pic13`	Produces position-independent code.
`+u1`	`-xmemalign=1i`	Specify the maximum assumed memory alignment and behavior of misaligned data accesses. Assume at most 1 byte alignment.

[1] On Oracle Solaris 10, -fast is a macro whose expansion varies from release to release, and platform to platform. A best practice is to place `–fast` first on the compilation command line as the right-most instance of an option overrides any number of previous instances.

`+u2`	`-memalign=2i`	Specify the maximum assumed memory alignment and behavior of misaligned data accesses. Assume at most 2 byte alignment.
`+DD[32][64]`	`-m[32][64]`	Specifies the memory model (32-bit or 64-bit) for the compiled binary object. On Oracle Solaris 10 –m32 is the default.
`+wsecurity`	`-errsecurity=v`	Check code for security loopholes. v={core|standard|extended|%none}
`+DSnative`	`-xtarget=native`	Generates code for native hardware.
`N/A`	`-nofstore`	Does not convert the value of a floating-point expression or function. Allows values to be kept in registers longer. (x86 only)
`-O`	`-O`	Equivalent to –x02.
`N/A`	`-xarch=`*`a`*	Limits the set of instructions the compiler uses to those for a particular processor architecture.
`+Oautopar` (+03 and above)	`-xautopar`	Automatically parallelizes for multiple processors.
`N/A`	`-xbuiltin [={%all|%none}]`	Improves the optimization of code that calls standard library functions. Lets the compiler substitute intrinsic functions or inline system functions where profitable for performance.
`N/A`	`-xcache=`*`c`*	Defines cache properties for the optimizer.
`N/A`	`-xchip=c`	Specifies the target processor for use by the optimizer.
`N/A`	`-xdepend`	Analyzes loops for inter-iteration data dependencies, and performs loop restructuring, including loop interchange and fusion, and scalar replacement.
`+Oautopar` `+Onoautopar (default)`	`-xexplicitpar` `-xautopar`	Turns on automatic parallelization for multiple processors.
`+Oinline=[`*`f1,…,fn`*`]`	`-xinline=[`*`f1,…,fn`*`]`	Inlines the listed functions.
`+Onoinline`	`-xinline=`	Does not inline functions.

`-ipo`	`-xipo`
	`-xcrossfile`
	`-xlinkopt`
		Enables interprocedural analysis.
		Enables analysis and inlining across different source files. (This option is more limited than `-xipo`, but highly effective when only key parts of a large application need to be considered as a unit.)
		Performs link time optimization.

`N/A`	`-xlibmieee`	Forces IEEE 754-style return values.
`+Olibcalls`	`-xlibmil`	Inlines some library routines.

`-N`	`-xMerge`	Merges data segments into text segments.
N/A	`-xnolibmil`	Does not inline math library routines.
`+O[0|1|2|3|4]`	`-xO[1|2|3|4|5]`	Specifies the optimization level.
N/A	`-xparallel`	Parallelizes loops automatically and as specified in the code.
`+O[no]dataprefetch`	`-xprefetch=[no%]auto,[no%]explicit`	Controls generation of data prefetch instructions.
N/A	`-xreduction`	Turns on reduction recognition during automatic parallelization.
N/A	`-xregs=`*r*	Specifies the usage of registers for generated code.
N/A	`-xrestrict=`*f*	Treats pointer-valued function parameters as restricted pointers.
N/A	`-xsafe=mem`	Allows the compiler to assume no memory protection violations occur.
`+Osize`	`-xspace`	Instructs the compiler not to perform optimizations or loop parallelizations that increase code size.
N/A	`-xtarget=`*t*	Specifies the target system for instruction set and optimization.
`+Oloop_unroll=`*n*	`-xunroll=`*n*	Specifies the loop unrolling level.
N/A	`-xvector[={yes|no}]`	Enables the automatic generation of vector calls for intrinsics. Requires -fround=nearest.
LINKING AND LIBRARY		

`-Wl`
`-a`
`-archive`
	`-Bstatic`	Searches static libraries.

`-Wl`
`-a`
`shared_archive`
	`-Bdynamic`	Searches dynamic libraries.

`-Wl, -dynamic`	`-dy`	Specifies dynamic linking (default).
`-Wl, -noshared`	`-dn`	Specifies static linking.
`-l`*name*	`-l`*name*	Loads a library.
N/A	`-mc`	Removes duplicate strings from the `.comment` section.
N/A	`-mr[,string]`	Removes all strings from the `.comment` section and inserts the specified string.

`-mt`	`-mt`	Instructs the compiler to compile and link multithreaded code using the Oracles Solaris threads or POSIX threads API.
`-Wl, +b,dir[:dir]`	`-Rdir[:dir]`	Specifies the runtime library search path.
N/A	`-xF`	Allows for optimal reordering of functions and variables by the linker. Requires use of `analyzer(1)`.
N/A	`-xlic_lib=1` `-xlic_lib=sunperf`	Links with a licensed Oracle library. Links to the Sun Performance Library.
`+Olit=[all|const]` (default)	`-features=` `conststrings`	Inserts string literals into the text segment.
`Use ld –b`	`-G`	Creates a shared object.
Use `ld +h name`	`-h name`	Assigns a name to a shared dynamic library (supports different names for libraries for versioning).
`-Ldir`	`-Ldir`	Adds a specified directory to the list of directories the link-editor uses to search for libraries.
N/A	`-xnolib`	Does not link any libraries by default (no `–l` options are passed to the link-editor, `ld`).
META OPTIONS		
None	`-###`	Shows each component as it would be invoked, but does execute it.
`-C`	`-C`	Prevents the preprocessor from removing comments.
`Dname [=tokens]`	`Dname [=tokens]`	Defines a symbol.
`-E`	`-E`	Runs the source file through the preprocessor.
`-wn`	`-erroff=t`	Suppresses `compiler` warnings.
N/A	`-errtags=[yes|no]`	Shows message tags.
`-Wp, -h`	`-H`	Prints header files.
`-Wl +s` (default)	`-i`	Passes an option to the linker instructing it to ignore the `LD_LIBRARY_PATH` and `LD_LIBRARY_PATH_64` environment variable settings.
N/A	`-keeptmp`	Retains temporary files created during compilation instead of deleting them automatically.
`-MO`	`-fd`	Reports K&R-style function definitions and declarations.
`+help`	`-flags`	Prints a summary of each compiler option.

N/A	`-noqueue`	Does not queue if a license is not available.
`-P`	`-P`	Runs the source file through the C preprocessor only.
N/A	`-Q[y][n]`	Emits or does not emit identification information to the output file. The default is `y`.
`-S`	`-S`	Directs the compiler to produce an assembly source file, but not assemble the program.
`-s`	`-s`	Removes symbolic debugging information.
`-U`*`name`*	`-U`*`name`*	Undefines a specified preprocessor symbol.
`-V`	`-V`	Directs the compiler to print the name and version ID of each component as the compiler executes.
N/A	`-v`	Directs the compiler to perform stricter semantic checks and to enable other `lint`-like checks.
`-W`*`c,arg`*	`-W`*`c,arg`*	Passes an argument to a specified component.
`-w`	`-w`	Suppresses compiler warning messages.
N/A	`-xCC`	Instructs the compiler to accept C++ style comments.
Default	`Default` `-xchar=[signed|s]`	Specifies that unqualified chars are signed.
`+uc`	`-xchar=[unsigned|u]`	Specifies that unqualified chars are unsigned.
`+help`	`-xhelp=f`	Displays online help information.
`+Oreport=loop`	`-xloopinfo`	Shows which loops are parallelized and which are not parallelized.
`Use makedepend(1)`	`-xM[1]`	Runs only the preprocessor and generates makefile dependencies.
N/A	`-maxopt=`*`n`*	Limits the level of `#pragma` opt.
`+time`	`-xtime`	Reports the time and resources used by each compilation component.
N/A	`-Yc,` *`dir`*	Specifies a new directory `dir` for the location of component `c`.
N/A	`-YA,` *`dir`*	Specifies a directory in which to search for compiler components.
N/A	`-xvpara`	Warns about loops that contain `#pragma` MP directives.

FILE HANDLING		
`-c`	`-c`	Produces a `.o` file only.
`-Idir`	`-Idir`	Specifies an include file.
`-o file`	`-o file`	Sets the output filename.
`-Idir`	`-YI, dir`	Changes the default directory searched for `include` files.
`-Ldir`	`-YP, dir`	Changes the default directory for finding library files.
`-Ldir`	`-YS, dir`	Changes the default directory for startup object files.
PERFORMANCE ANALYSIS AND DEBUG		
`N/A`	`-xinstrument=datarace`	Instruments code for data race conditions (Thread Analyzer).
`-g`	`-g`	Generates debug information.
`-p`	`-p, -qp`	Produces additional symbol table information for debugging.
`N/A`	`-xa`	Instruments code for test coverage with `tcov(1)`.
`-G`	`-xpg`	Prepares the object code to collect data for profiling with `gprof(1)`.
`+Oprofile=collect`	`-xprofile=collect`	Collects data for a profile or uses a profile to optimize.
`+Oprofile=usefilename`	`-xprofile=use[:name]`	Uses execution frequency data collected from code compiled with `-xprofile=collect` to optimize for the work performed when the profiled code was executed.
`N/A`	`-xprofile=tcov`	Instruments object files for basic block coverage analysis using `tcov(1)`.
`N/A`	`-xs`	Allows debugging by `dbx` without object files.
`N/A`	-xhwcprof	Enables compiler support for hardware counter-based profiling.
C COMPILER-SPECIFIC OPTIONS		
`N/A`	`-xsfpconst`	Represents unsuffixed floating-point constants as single precision.
`N/A`	`-Aname [(tokens)}`	Associates a name with tokens.
`-A[a|e]` `-Aa`	`-X[a|c|s|t]` `-Xc (strictest ANSI)`	Specifies the language dialect (K&R through strict ANSI).
`N/A`	`-xP`	Prints prototypes for all K&R C functions.

N/A	`-xtransition`	Issues warnings for K&R C and Oracle Solaris ANSI C differences.
Use `$TMPDIR`	`-xtemp=dir` `export TMPDIR`	Sets the directory for temporary files used by the compiler (overrides `TMPDIR`).
N/A	`-Zll`	Creates the program database for `lock_lint`, providing static analysis of parallelization coding errors.
C++ COMPILER-SPECIFIC OPTIONS		
Use `$TMPDIR`	`-temp=dir` `export TMPDIR`	Sets the temporary directory (overrides the `TMPDIR` environment variable).
`+d`	`+d`	Prevents the compiler from expanding inline functions.
N/A	`-features`	Enables/disables various C++ language features.
N/A	`-inline=rlst`	Instructs the compiler to inline specified functions.
N/A	`-instance=a`	Controls template instances.
Use `-l`*l*	`-library=`*l*`[,…`*l*`]`	Loads CC libraries.

`-D_POSIX_C_SOURCE_199506L`
`-D_REENTRANT`
`-lpthread`
	`-mt`	Instructs the compiler to compile and link multithreaded code using the Oracles Solaris threads or POSIX threads API.

`-noeh`	`-features=no%except`	Does not generate code that supports C++ exceptions.
`-DNDEBUG`	`+p`	Disables assert statements.
N/A	`-template=wholeclass`	Instantiates whole template classes.
`-I`*path*	`-Ipath`	Specifies the search directory for template source, `-pti`*path* can be used, but complicates search rules and is not recommended.
`+inst_v`	`-verbose=template`	Controls template verbosity.
`+We` *nnn*`[,`*nnn*`]`	`-xwe`	Converts all warnings to errors.

Building Applications

The tools for building applications—link-editors (linkers), run-time link editing tools, shared libraries, and make tools—are significantly different on HP-UX and Oracle Solaris. This section provides an overview of the facilities on Oracle Solaris, and highlights the differences between the platforms.

Overview of Linking Concepts in Oracle Solaris

In Oracle Solaris, developers create applications and libraries using the link-editor, `ld(1)`, and execute these objects with the aid of the runtime linker `ld.so.1(1)`. The link-editor concatenates and interprets data from one or more input files. These files can be relocatable objects, shared objects, or archive libraries. From these input files, one output file is created. This file is a relocatable object, an executable application, or a shared object. The link-editor is most commonly invoked as part of the compilation environment, and works with files that conform to the executable and linking format (ELF).

On HP-UX and Oracle Solaris, the runtime linker, `ld.so.1(1)`, processes dynamic executables and shared objects at runtime, binding the executable and shared objects together to create a runnable process. During the generation of these objects by the link-editor, appropriate bookkeeping information is produced to represent the verified binding requirements. This information enables the runtime linker to load, relocate, and complete the binding process.

During process execution, the facilities of the runtime linker are made available. These facilities can be used to extend process address space by adding additional shared objects on demand. The two most common components involved in runtime linking are dynamic executables and shared objects. *Dynamic executables* are applications that are executed under the control of a runtime linker. These applications usually have dependencies in the form of shared objects that are located and bound by the runtime linker to create a runnable process. *Shared objects*, often referred to as *shared libraries*, provide the key building-block to a dynamically linked system. Similar to the HP-UX environment, shared objects are not lazy loaded in Oracle Solaris unless otherwise specified.

Lazy Loading of Dynamic Dependencies

Every dynamic object is examined for dependencies when loaded into memory. Identified dependencies are loaded immediately by default. After the entire dependency tree is analyzed, inter-object data references that are specified by relocations are resolved. These operations are performed whether or not the code in these dependencies is referenced by the application during its execution.

Lazy loading enables the loading of a dependency to be delayed until the function is first referenced. In such a scheme, unreferenced objects are never loaded. The HP-UX and Oracle Solaris linkers both provide an option to enable lazy loading. Alternatively the `dlopen()` and `dlsym()` functions can be used to load and bind to a dependency when needed. This model is ideal if the number of references is small, and works well if the dependency name or location is not known at link-edit time. For more complex interactions with known dependencies, coding to normal symbol references and designating the dependency to be lazily loaded is simpler.

On Oracle Solaris, an object is designated as lazily or normally loaded through the link-editor options `-z lazyload` and `-z nolazyload` respectfully. These options are position-dependent on the link-edit command line. Any dependency that follows the option takes on the loading attribute specified by the option. By default, the `-z nolazyload` option is in effect.

Direct Binding

Objects that use direct binding maintain the relationship between a symbol reference and the dependency that provided the definition. The runtime linker uses this information to search directly for the symbol in the associated object, rather than carry out the default symbol search model. Direct binding information for a dynamic object is recorded at link-edit time. This information can be established only for the dependencies that are specified with the link-edit of that object. Use the `–z defs` option to ensure all necessary dependencies are provided as part of the link-edit process.

The direct binding of a symbol reference to a symbol definition can be established with the `–B direct` option. This option establishes direct bindings between the object being built and all of the object's dependencies, as well as between any symbol reference and symbol definition within the object being built. Note that the `-B direct` option also enables lazy loading (equivalent to adding `–z lazyload` to the front of the link-edit command line).

Runtime Linking Functions

The runtime linker on HP_UX and Oracle Solaris, `ld.so.1`, provides several library calls that can be used to locate and bind applications to shared libraries during execution.

TABLE 6-3. SUPPORTED CALLS FOR DYNAMIC RUNTIME LINKING

DESCRIPTION	FUNCTION
Translates an address to symbolic information.	`dladdr`
Closes a shared object and unloads it.	`dlclose`
Returns the last error that occurred during the dynamic linking process.	`dlerror` (`dlerrno` on HP-UX)
Makes an executable object file available to a running process.	`dlopen`
Gets the address of a symbol in a shared object or executable.	`dlsym`

Mapfiles

The HP-UX and Oracle Solaris link-editor automatically and intelligently map input sections from relocatable objects to segments in the output file. On Oracle Solaris, developers can change the default mapping through the use of the `–m` option and a mapfile. The mapfile enables new segments to be created, attributes to be modified, and symbol versioning information to be supplied. Mapfiles consist of a series of directives, including:

- *Segment declarations* create a new segment in the output file, or change the attribute values of an existing segment
- *Mapping directives* instruct the link-editor on how to map input sections to output segments
- *Section-to-segment ordering* specifies the order in which sections are placed within a segment
- *Size-symbol declarations* enable developers to define a new global-absolute symbol that represents the size, in bytes, of the specified segment
- *File control directives* specify which version definitions within shared objects are to be made available during a link-edit

More information on mapfiles can be found in the *Linker and Libraries Guide*. Sample mapfiles can be found in the `/usr/lib/ld` directory.

Support Tools

Oracle Solaris provides similar support tools to HP-UX for the analysis and inspection of objects and linking processes. Table 6-4 lists the key support tools on each platform.

TABLE 6-4. COMPARISON OF SUPPORT TOOLS AND LIBRARIES

DESCRIPTION	HP-UX	ORACLE SOLARIS 10
Displays or modifies internal object file attributes.	`chatr(1)`	–
Dumps select portions of an object file.	–	`dump(1)`
Contains the object file access library.	`elf(3E)`	`elf(3ELF)`
Displays the contents of an ELF file.	`elfdump(1)`	`elfdump(1)`
Writes binding information into an executable.	`fastbind(1)`	
Analyzes the interface requirements of dynamic ELF objects.	–	`lari(1)`
Lists dynamic dependencies specified at runtime.	`ldd(1)`	`ldd(1)`
Finds ordering relation for an object or library archive.	`lorder(1)`	`lorder(1)`
Displays the symbol table for an ELF object file.	`nm(1)`	`nm(1)`

Lists the dynamic libraries linked into each process, including shared objects explicitly attached using `dlopen(3C)`.	`pldd(1)`	`pldd(1)`
Prints a hex+symbolic stack trace for each process or specified lightweight processes in each process.	`pstack(1)`	`pstack(1)`
Displays internal version information contained in an ELF file.	—	`pvs(1)`
Produces segment or section size information in bytes for each loaded section in ELF object files.	`size(1)`	`size(1)`
Strips symbol table, debugging, and line number information from an object file.	`strip(1)`	`strip(1)`

Environment Variables

The HP-UX and Oracle Solaris link-editors support a number of environment variables that begin with the characters `LD_`, such as `LD_LIBRARY_PATH`. On Oracle Solaris, each environment variable can exist in its generic form, or can be specified with a `_32` or `_64` suffix to make it specific to a 32-bit or 64-bit process. This suffix also overrides any generic, non-suffixed, version of the environment variable that might be in effect. Table 6-5 lists the key linker environment variables.

TABLE 6-5. KEY LINKER ENVIRONMENT VARIABLES

DESCRIPTION	ORACLE SOLARIS 10
Causes the runtime linker to perform immediate reference and lazy reference relocations when an object is loaded.	`LD_BIND_NOW`
Specifies an alternative configuration file.	`LD_CONFIG`
Enables debugging.	`LD_DEBUG`
Specifies an output file for use instead of the standard error.	`LD_DEBUG_OUTPUT`
Specifies the library search path.	`LD_LIBRARY_PATH`
Disables direct bindings.	`LD_NODIRECT`
Disables lazy loading.	`LD_NO_LAZYLOAD`
Defines options to be used by the linker.	`LD_OPTIONS`

Specifying Link Editor Options

Most options to the link-editor can be passed through the compiler driver command line or Oracle Solaris Studio. For the most part, the compiler and the link-editor options do not conflict. Where a conflict arises, use a command line option to pass specific options to the link-editor, or set the `LD_OPTIONS` environment variable.

When porting applications from HP-UX to Oracle Solaris, closely examine the link options specified in existing makefiles and compare them to the options provided by the linker on Oracle Solaris. While several options are identical in the two environments, most options are named or behave differently on the two platforms. Take care to translate linker options correctly. Table 6-6 identifies a few linker options and summarizes their behavior on HP-UX and Oracle Solaris. See the *Linker and Libraries Guide* for detailed information on all available linker options.

Applications that require numerous linking options can create long and complicated command lines. In the HP-UX and Oracle Solaris environments, options can be directed to the linker from a file. Annotations can be made in the options file through the use of comments, lines that begin with the # character.

TABLE 6-6. HP-UX AND ORACLE SOLARIS KEY LINKER OPTION COMPARISON

OPTION	ORACLE SOLARIS DESCRIPTION	NOTES ON BEHAVIORAL DIFFERENCES ON HP-UX
`-64`	Creates a 64-bit object.	

`-B direct|nodirect`
`dynamic|static`
`eliminate`
`group`
`local`
`reduce`
`symbolic`
	Controls binding behavior.	HP arguments differ:
		`deferred, immediate, nodelete, nonfatal, restricted, symbolic, verbose`

`-D` *`tokens`*	Prints debugging information.	
`-F` *`filename`*	Identifies *filename* as a filter for the shared object.	HP option is named `+filter`
`-G`	Produces a shared library.	HP option strips debug information
`-L`*`directory`*	Adds *directory* to the library search path.	
`-M` *`mapfile`*	Specifies a mapfile to use to alter default mapping.	HP option is named `-k`
`-R` *`arg`*	Specifies a list of library directories for the runtime linker.	HP option performs a different function. It defines the origin of a text segment.

Makefiles

Oracle Solaris provides the standard `make(1)` command that is also available on HP-UX, as well as the Distributed Make utility, `dmake(1)`, a superset of `make(1)`. Integrated into the Oracle Solaris Studio IDE, `dmake` parses makefiles and determines which targets can be built concurrently, and distributes the build of those targets over a number of hosts set by the developer. Makefiles that use the standard `make` utility on HP-UX require little alteration when moved to `make` or `dmake` on Oracle Solaris. With nested makes, if a top-level makefile calls `make`, use `$(MAKE)`.

The `dmake` command is executed on a *dmake host* and *jobs* are distributed to *build servers*. Jobs are distributed based on the makefile targets that `dmake` determines (based on makefiles) can be built concurrently. Any system can be a build server as long as the `rsh` command can be executed without requiring a password, and the system can access the bin directory in which the `dmake` software is installed. From the *dmake host* developers can control which build servers are used and how many jobs are allotted to each build server.

Comparison of Makefile Attributes

The standard `make` utility provided on HP-UX and Oracle Solaris, and the `dmake` utility, support most of the same suffixes, macros, and options (Table 6-7).

TABLE 6-7. MAKEFILE SUMMARY

ITEMS THAT ARE THE SAME FOR MAKE (HP-UX AND ORACLE SOLARIS) AND DMAKE					
BUILT-IN TARGETS	**OPTIONS**		**VARIABLES**		**DYNAMIC MACROS**
`.DEFAULT`	`-d`	`-p`	`CC`	`LD`	`$@`
`.IGNORE`	`-e`	`-q`	`CFLAGS`	`LDLAGS`	`$%`
`.PRECIOUS`	`-f makefile`	`-r`	`CPPFLAGS`	`LEX`	`$*`
`.SILENT`	`-i`	`-s`	`FC`	`LFLAGS`	`$<`
`.SUFFIXES`	`-k`	`-S`	`FFLAGS`	`YACC`	`$?`
	`-n`	`-t`		`YFLAGS`	

ADDITIONAL OPTIONS FOR DMAKE		
OPTION	**ARGUMENTS**	**DESCRIPTION**
`-c`	`dmake_rcfile`	Specifies an alternate runtime configuration file.
`-g`	`dmake_group`	Specifies the name of the build server group for jobs distribution.
`-j`	`dmake_max_jobs`	Specifies the maximum number of jobs that are distributed to the group of build servers in the runtime configuration file.
`-m`	`serial|parallel|distributed`	`serial`: dmake behaves like standard serial make. `parallel`: dmake distribute jobs only to the dmake host. `distributed`: dmake behaves in fully distributed mode (default).
`-o`	`dmake_odir`	Specifies a physical directory for temporary files.

Debugging Applications

Oracle Solaris Studio provides a rich set of integrated debugging tools that can help developers understand how applications are behaving, and why, and shorten the development/testing cycle.

Discover

The Sun Memory Error Discovery Tool software detects memory access errors. The tool works with any binary—even a fully optimized binary—that has been prepared by the compiler. The binary is instrumented with a single command and run in the normal way. Memory access errors are caught and reported dynamically as the program executes. The Sun Memory Error Discovery Tool reports the memory anomalies found during the run in a text or HTML file. Note that the tool operates on executed code. If a portion of user code is not executed at run time, errors in that portion are not reported.

Uncover

The Oracle Solaris Studio Code Coverage Tool measures code coverage of applications. An *uncoverage* feature makes it possible to quickly find major functional areas within binaries that are not being tested. The coverage information reported can be at a function, statement, basic block, or instruction level.

The Code Coverage Tool is multithread and multiprocess safe, and uses a simple procedure for instrumenting binaries, running tests, and displaying results. Special builds are not required, making it easy for developers to use the tool with regular or optimized binaries that are ready for production. Overhead is slight, and the performance impact of the tool relative to uninstrumented code is fairly small.

The dbx Debugger and the GUI-Based dbxtool

The scriptable dbx debugger is multithread-aware and can debug multithreaded applications that use either Oracle Solaris or POSIX threads. Developers can examine stack traces of each thread, resume all threads, step through or over a specific thread, and navigate between threads. The debugger also supports fixing and continuing, the process of relinking source files after making changes without having to recompile the entire program or restart the debugging session. By eliminating the time consuming steps of rebuilding the program and starting the debugger again from the beginning, developers can get programs working more rapidly. In addition, dbx can help track down difficult bugs by providing developers with valuable memory information including usage, leaks, and what is being accessed.

Runtime Checking

Runtime checking is an integral debugging feature of Oracle Solaris Studio that automatically detects runtime errors such as memory access and memory leaks in an application during the development phase. As errors are detected, the debugger interrupts program execution and displays the relevant source code, enabling bugs to be fixed as they are found.

With support for multithreaded code, runtime checking provides a valuable means of debugging more complex multithreaded applications. Oracle Solaris Studio runtime checking works with all languages and requires no recompiling, relinking, or makefile changes. All debugging operations can be performed while using runtime checking except collecting performance data.

Oracle Solaris DTrace Facility and DLight

DTrace is a comprehensive dynamic tracing facility built into Oracle Solaris that gives users, administrators, and developers a new and unique level of observability into the behavior of user programs and the operating system. Hundreds of thousands of tracing points, or probes, are embedded in the Oracle Solaris kernel, utilities, and other software components to enable dynamic instrumentation of user-specified probes for recording data and examining the system in-depth. Trace points are completely passive until enabled for data collection, and can be disabled when observation no longer is required. Observed information can help developers to rapidly identify performance bottlenecks, optimize resource utilization and performance, and quantify application resource requirements.

DLight is a new GUI tool with a simple drag-and-drop interface that utilizes and visualizes the power of DTrace debugging and performance analysis functionality. Developers can select the target application, choose the DLight instrumentation to be monitored while the target application is running, and analyze the data returned by the tool. The data returned can be used to refine the experiment recursively until the behavior of the application under analysis is clear. The DLight tool can be applied to an attached process or an executable.

Optimizing Applications

Maximizing application performance is a key goal for any optimizing compiler technology. However, modern application performance must be seen in the context of a diverse and complex mixture of heterogenous hardware and operating systems, as well as and both serial and parallel environments. For example, the latest x86 processors from both Intel® and AMD™ now implement Streaming SIMD Extensions 2 (SSE2) supplemental instructions while some SPARC processors support special instructions that can dramatically increase performance for certain kinds of operations. In addition, all major chip vendors are now producing multicore CPUs, including Intel® Xeon®, AMD Opteron, and Oracle SPARC processors.

Optimizing for Serial Performance

Getting the best performance for SPARC or x86 applications involves using the latest compilers and selecting the best and most appropriate set of compiler options. The sections that follow detail a number of recommended options for optimizing applications for serial performance. Optimizing multithreaded or parallel applications is covered later in this document.

Oracle Solaris Studio compilers strive to provide the best out-of-the-box performance for any applications built using them. However, it is often the case that some minor refinements to the selection of compiler options can yield further gains in performance. As a result, it is key that optimization and tuning be approached on an experimental basis before the final version of the program is released. As a part of this process, it is key to understand exactly what is expected of the compiler in concert with the assumptions made in the application. In particular, two key questions must be asked when selecting appropriate compiler options:

- What is known about the platforms where the compiled application will eventually run?
- What is known about the assumptions that are made in the code?

In addition, it is helpful to consider the purpose of a particular compilation. Compiler options can present various trade-offs depending on whether a given compilation is meant to assist with debugging, testing, tuning, or final performance optimization.

Identifying the Target Platform

Knowing where the code will eventually run is essential in order to understand what optimization options make sense. The choice of platform determines:

- A 32-bit or 64-bit instruction set
- Instruction set extensions the compiler can use to accelerate performance
- Instruction scheduling depending on instruction execution times
- Cache configuration

Generating 32-bit or 64-bit Code

The UltraSPARC and x86 processor families can run both 32-bit and 64-bit code. The principal advantage of 64-bit code is that the application can handle a larger data set than 32-bit code. However, the cost of this larger address space is a larger memory footprint for the application, since long variable types and pointers increase in size from 4 bytes to 8 bytes. The increase in memory footprint might cause a 64-bit version of an application to run more slowly than the 32-bit version.

At the same time, the x86 platform presents some architectural advantages when running 64-bit code as compared to running 32-bit code. In particular, the application can use more registers, and can use a better calling convention. On the x86 platform, these advantages will typically allow a 64-bit version of an application to run faster than a 32-bit version of the same code, unless the memory footprint of the application has significantly increased.

The UltraSPARC line of processors took a different approach, as it was architected to enable a 32-bit version of an application to use the architectural features of the 64-bit instruction set. As a result, there is no architectural performance gain going from 32-bit to 64-bit code. Consequently, 64-bit applications compiled for UltraSPARC processors only see the additional cost of the increase in memory footprint.

Compiler flags determine whether a 32-bit or 64-bit binary is generated.

- The `-m32` flag generates a 32-bit binary
- The `-m64` flag generates a 64-bit binary

For additional details about migrating from 32-bit to 64-bit code, refer to "Converting 32-bit Applications Into 64-bit Applications: Things to Consider" at http://www.oracle.com/technetwork/server-storage/solaris/ilp32tolp64issues-137107.html and "64-bit x86 Migration, Debugging, and Tuning with the Sun Studio 10 Toolset" at http://www.oracle.com/technetwork/server-storage/solaris/amd64-migration-137808.html.

Specifying an Appropriate Target Processor

Oracle Solaris Studio compilers allow considerable flexibility in selecting a target processor through setting the `-xtarget` compiler flag. The default for the compiler is to produce a "generic" binary – namely a binary that will work well on all platforms (`-xtarget=generic`). In many situations, a generic binary will be the best choice. However, there are some situations where it is appropriate to select a different target, including:

- **To override a previous target setting.** The compiler evaluates options from left to right, and if the flag `-fast` has been specified on the compile line, then it may be appropriate to override the implicit setting of `-xtarget=native` with a different choice.
- **To exploit the features of a particular processor.** For example, newer processors tend to have more features that can be exploited for performance gains. The compiler can use these features at the expense of producing a binary that does not run on older processors that do not have these features.

The `-xtarget` flag actually sets three flags:

- The `-xarch` flag specifies the architecture of the target machine. This architecture is basically the instruction set that the compiler can use. If the processor that runs the application does not support the appropriate architecture then the application may not run.
- The `-xchip` flag tells the compiler which processor to assume is running the code. This flag tells the compiler which patterns of instructions to favor when it has a choice between multiple ways of coding the same operation. It also tells the compiler the instruction latency to use in order that the instructions are scheduled to minimize stalls.
- The `-xcache` flag tells the compiler the cache hierarchy to assume. This selection can have a significant impact on floating point codes where the compiler is able to make a choice about how to arrange loops so that the data being manipulated fits into the caches.

Target Architectures for the SPARC® Processor Family

For the SPARC processor family, the default setting `-xtarget=generic` should be appropriate for most situations. This setting will generate a 32-bit binary that uses the SPARC V8 instruction set, or a 64-bit binary that uses the SPARC V9 instruction set. The most common situation where the target architecture needs to be taken into account and a different setting may be required is compiling code that contains significant floating point computations.

Target Architectures for the x86 Processor Family

By default, the Oracle Solaris Studio compiler targets a 32-bit generic x86 based processor, so that generated code will run on any x86 processor from a Pentium Pro to the latest Intel or AMD Opteron processor. While `-xtarget=generic` produces code that can run over the widest range of processors, this code will not take advantage of the SSE2 extensions offered by the latest processors. To exploit these instructions, the flag `-xarch=sse2` can be used. However, the compiler may not recognize all opportunities to use these instructions unless the vectorization flag `-xvector=simd` is also used.

Table 6-8 provides a summary of Oracle Solaris Studio compiler flags recommended for compilation for various SPARC and x86 target architectures.

TABLE 6-8. ORACLE SOLARIS STUDIO FLAGS FOR SPECIFYING ARCHITECTURE AND ADDRESS SPACE

ARCHITECTURE	32-BIT ADDRESS SPACE	64-BIT ADDRESS SPACE
SPARC	`-xtarget=generic -m32`	`-xtarget=generic -m64`
SPARC64	`-xtarget=sparc64vi -m32`	`-xtarget=sparc64vi -m64`
x86	`-xtarget=generic -m32`	`-xtarget=generic -m64`
X86/SSE2	`-xtarget=generic -xarch=sse2 -m32–xvector=simd`	`-xtarget=generic -xarchsse2 -m64 -xvector=simd`

Choosing Compiler Optimization Options

Choosing compiler options resents a trade-off between compilation time, run-time, and (possibly) application behavior. The optimization flags chosen alter three important characteristics, including:

- The runtime of the compiled application
- The length of time that the compilation takes
- The amount of debug activity that is possible with the final binary.

In general, the higher the level of optimization, the faster the application runs (and the longer it takes to compile), but the less debug information that is available. Ultimately, the particular impact of optimization levels will vary from application to application. The easiest way of thinking about these tradeoffs is to consider three degrees of optimization, as outlined in Table 6-9.

TABLE 6-9. THREE DEGREES OF OPTIMIZATION GENERATE DIFFERENT IMPLICATIONS FOR RESULTING CODE.

PURPOSE	FLAGS	COMMENTS
Full debug	`-g` [no optimization flags]	The application will have full debug capabilities, but no optimization will be performed on the application, leading to lower performance.
Optimized	`-g -O` [`-g0` for C++]	The application will have good debug capabilities, and a reasonable set of optimizations will be performed on the application, typically leading to significantly better performance.
High Optimization	`-fast` [`-g0` for C++] `-g`	The application will have good debug capabilities, and a large set of optimizations will be performed on the application, typically leading to higher performance.

Compiling for Debugging (`-g`)

The `-g` option is a high-fidelity debug option that lets the developer check for algorithmic error. With the flag set, code performs exactly as written and the developer can inspect variables under the debugger. For lower levels of optimization, the `-g` flag disables some minor optimizations (to make the generated code easier to debug). At higher levels of optimization, the presence of the flag does not alter the code generated (or its performance). However, it is important to be aware that at high levels of optimization, it is not always possible for the debugger to relate the disassembled code to the exact line of source, or for it to determine the value of local variables held in registers rather than stored to memory.

The C++ compiler will disable some of the inlining performed by the compiler when the `-g` compiler flag is used. For C++, the `-g0` flag will tell the compiler to do all the inlining that it would normally perform, as well generating the debug information.

A very strong reason for compiling with the `-g` flag is that the Oracle Solaris Studio Performance Analyzer can then attribute time spent in the code directly to lines of source code – making the process of finding performance bottlenecks considerably easier.

Basic Optimization (-O)

Basic optimization can be achieved by using the `-O` compiler flag. The `-O` flag offers decent runtime performance, without taking excessively long to compile the application. The `-g` flag can be added to the `-O` flag to get optimization with debugging information built in. Multiple possible levels of optimization are offered with Oracle Solaris Studio compilers, including `-O3`, `-O4`, and `-O5`. Please see the Oracle Solaris Studio documentation for a full description of these options.

Aggressive Optimization (-fast)

The `-fast` option is a good starting point when optimizing code, but it may not necessarily represent the desired optimizations for the finished program. Developers should note that because the `-fast` option is defined as a particular selection of compiler options, it is subject to change from one release to another, as well as between compilers. In addition, some of the component options selected by `-fast` may not be available on some platforms. Care must also be taken if application compilation and linking are performed separately. Developers should make sure that applications are both compiled and linked with `-fast` to ensure proper behavior.

The `-fast` option implies many individual compilation optimizations. These individual options can be turned off or on at will. Ideally the `-fast` option should be applied objectively. For instance, if compiling with `-fast` yields a five-fold performance gain, it is definitely worth exploring which of the specific options included in `-fast` are providing the performance advantages. Those options might then be used individually in subsequent builds for a more deterministic and focused optimization.

Note: The `–fast` compilation flag can have implications for target architecture, floating-point arithmetic, and pointer aliasing. Please see the white paper "Optimizing Applications with Oracle Solaris Studio Compilers and Tools" located at http://www.oracle.com/technetwork/systems/optimizing-apps-oracle-solaris-stud-150254.pdf for more information.

Performance Analyzer

As computer systems continue to become more powerful, application performance is emerging as a critical factor, with bad performance increasingly considered a program failure. Developers are now keenly aware that they must streamline critical sections of source code as well as locate programmatic errors and coding deficiencies without impacting application accuracy. Oracle Solaris Studio includes a Performance Analyzer that can help aid developers with these tasks.

To use the Performance Analyzer, applications can be compiled with any level of parallelization and optimization. To see source code, and to attribute time to lines of source code, the `-g` option must also be specified. Applications are then run using the `collect` command. The command can specify a PID,

```
% collect –P <pid>
```

or the collect command can be used to launch the application and its parameters.

```
% collect <application> <parameters>
```

The collect command gathers performance data during application execution, saving it to an experiment file to be used later during the analysis process. The `collect` command enables developers to obtain information on:

- Clock-based profiles
- Thread-synchronization delay events and wait time
- Operating system summary information
- Hardware-counter overflow profiles on systems where the hardware supports it
- Global information, including execution statistics and address-space data

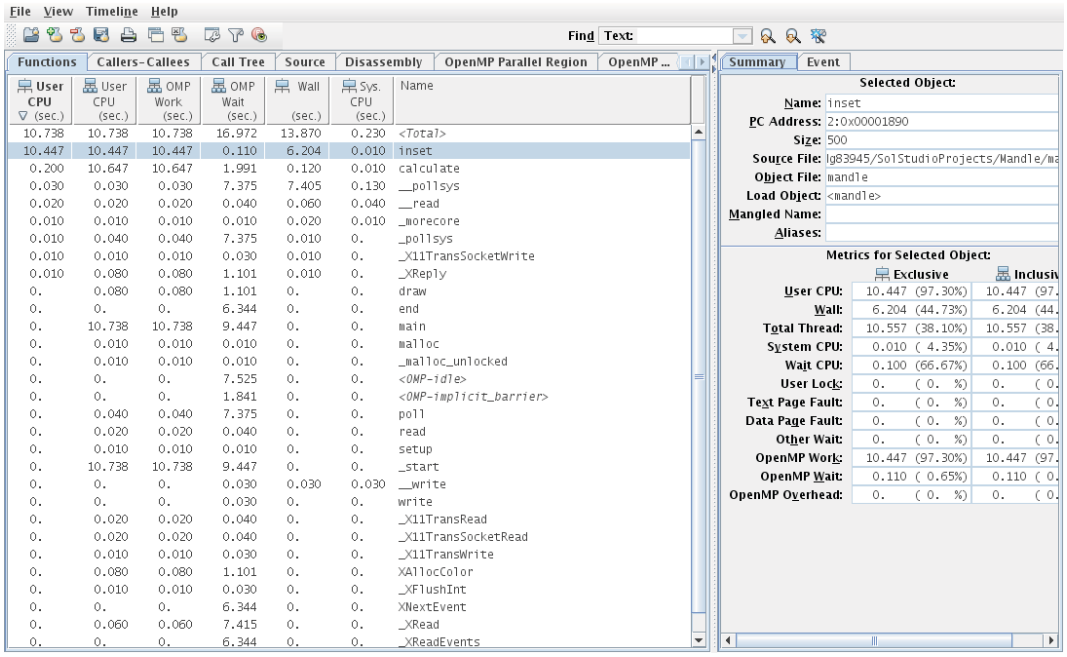
Once the experiment is complete, the Performance Analyzer loads the experiment data from a file titled `test.1.er`. Experiments can be either loaded into the analyzer from the command line, or by using the <*File*> menu from the running analyzer application. To start the analyzer, the following is typed on the command line.

```
% analyzer <control-options> <experiment-list>
```

To aid application analysis, the Performance Analyzer then provides several ways for developers to view collected performance data, including data display at the function or load object level. Developers can control which metrics are shown, as well as the order in which they appear. The Performance Analyzer features multiple tabs which provide different perspectives into the runtime environment, including:

- The Functions tab
- The Callers-Callees tab
- The Disassembly tab
- The Source tab
- The Timeline tab

The Functions Tab (Figure 6-3) shows a list of functions and their metrics. The metrics are derived from the data collected in the experiment. Metrics can be either exclusive or inclusive. Exclusive metrics represent usage within the function itself while inclusive metrics represent usage within the function, and all of the functions it called.



User CPU (sec.)	User CPU (sec.)	OMP Work (sec.)	OMP Wait (sec.)	Wall (sec.)	Sys. CPU (sec.)	Name
10.738	10.738	10.738	16.972	13.870	0.230	<Total>
10.447	10.447	10.447	0.110	6.204	0.010	inset
0.200	10.647	10.647	1.991	0.120	0.010	calculate
0.030	0.030	0.030	7.375	7.405	0.130	__pollsys
0.020	0.020	0.020	0.040	0.060	0.040	__read
0.010	0.010	0.010	0.010	0.020	0.010	_morecore
0.010	0.040	0.040	7.375	0.010	0.	_pollsys
0.010	0.010	0.010	0.030	0.010	0.	_X11TransSocketWrite
0.010	0.080	0.080	1.101	0.010	0.	_XReply
0.	0.080	0.080	1.101	0.	0.	draw
0.	0.	0.	6.344	0.	0.	end
0.	10.738	10.738	9.447	0.	0.	main
0.	0.010	0.010	0.010	0.	0.	malloc
0.	0.010	0.010	0.010	0.	0.	_malloc_unlocked
0.	0.	0.	7.525	0.	0.	<OMP-idle>
0.	0.	0.	1.841	0.	0.	<OMP-implicit_barrier>
0.	0.040	0.040	7.375	0.	0.	poll
0.	0.020	0.020	0.040	0.	0.	read
0.	0.010	0.010	0.010	0.	0.	setup
0.	10.738	10.738	9.447	0.	0.	_start
0.	0.	0.	0.030	0.030	0.030	__write
0.	0.	0.	0.030	0.	0.	write
0.	0.020	0.020	0.040	0.	0.	_X11TransRead
0.	0.020	0.020	0.040	0.	0.	_X11TransSocketRead
0.	0.010	0.010	0.030	0.	0.	_X11TransWrite
0.	0.080	0.080	1.101	0.	0.	XAllocColor
0.	0.010	0.010	0.030	0.	0.	_XFlushInt
0.	0.	0.	6.344	0.	0.	XNextEvent
0.	0.060	0.060	7.415	0.	0.	_XRead
0.	0.	0.	6.344	0.	0.	_XReadEvents

Figure 6-3. The Performance Analyzer Function Tab lets developers understand where time is being spent.

Optimizing Parallel Applications

Most processors today—SPARC and x86 alike—are equipped with multiple cores and are capable of supporting multiple simultaneous execution threads. Many systems also employ multiple multicore processors. Taking advantage of these multiple cores and exploiting multiple threads of execution has become important as organizations seek to derive as much value and performance as possible from their selected platforms.

Oracle Solaris provides an efficient and scalable threading model as well as a smart scheduler to deliver these considerable resources to applications through a variety of application development and deployment tools.

- Virtualization systems such as Oracle VM and Oracle VM Server for SPARC let multiple operating system instances share a single physical system.
- Threaded Oracle Solaris Containers allow multiple execution environments within a single operating system instance.
- Threaded applications can take advantage of multiple cores on multicore processors and multisocket systems.

Independent of the execution environment, as developers seek to exploit parallelism, they must ensure that their code is correct and provides predictable results. Oracle Solaris Studio compilers support techniques for generating parallel applications, including automatic parallelization, support for OpenMP directives, and support for the POSIX threads API. The Oracle Solaris Studio Thread Analyzer is also provided to help analyze parallel code for correctness.

Automatic Parallelization

Many existing codes were written without the assumption of parallel threads of execution. Oracle Solaris Studio compilers provide mechanisms to let the application run multiple threads without requiring the developer to specific how. Loops in particular often represent opportunities where a previously repetitive serial operation can be divided into multiple independent execution threads. Several compiler flags are used with Oracle Solaris Studio compilers to govern automatic parallelization behavior.

- The `-xautopar` compiler flag tells the compiler to look for loops that can be safely parallelized in the code.
- The `-xreduction` compiler flag can be used to recognize and parallelize reduction operations that take a range of values and output a single value – such as summing all the values in an array.
- The `-xloopinfo` compiler flag can be specified to generate information for the developer about the loops that the compiler has parallelized.

OpenMP

Support for OpenMP in Oracle Solaris Studio means that the compilers can look for directives (pragma) in the source code in order to build a parallel version of the application. Similar to automatic parallelization, the compiler does the work so that developers do not have to manage their own threads. OpenMP represents an incremental approach to parallelization with potentially fine granularity. OpenMP allows developers to set directives around specific loops to be optimized through threading while leaving other loops untouched. The other distinct advantage of this approach is that developers can derive a serial and a parallel version of the application from the exact same code base, which can be helpful for debugging. Several compiler flags are used with Oracle Solaris Studio related to OpenMP.

- Enable OpenMP with the `-xopenmp` compiler flag. Directives are recognized only when the flag is used.
- Set the `-xvpara` compiler flag to report potential parallelization issues.
- Set the `-loopinfo` compiler flag to instruct the compiler to provide details on which loops were parallelized.
- Set the `OMP_NUM_THREADS` or `PARALLEL` environment variable at runtime to take advantage of multiprocessor execution. These environment variables specify the number of processors available to the program. The variables are equivalent, and only one needs to be set.

POSIX Pthreads

By programming to the POSIX threads API, developers can have complete control over thread usage in their applications. POSIX Threads (`pthreads`) represents a POSIX standard for a thread API, defining a set of C programming language types, functions, and constants. Oracle Solaris Studio compilers support the POSIX threads programming model.

Thread Analyzer

While the Performance Analyzer provides an advanced tool for application optimization, the Thread Analyzer is designed to help ensure multithreaded application correctness. Specifically, the Thread Analyzer can help detect, analyze, and debug the special situations that can arise in multithreaded applications.

- **Data races** can cause incorrect or unpredictable results, and can occur arbitrarily far way from where a problem seems to occur. Data races occur under the following conditions:
  - Two or more threads in a single process concurrently access the same memory location
  - At least one of the threads is accessing the memory location for writing
  - The threads are not using any exclusive locks to control their accesses to that memory
- **Deadlock conditions** occur when one thread is blocked waiting on a resource held by a second thread, while the second thread is blocked waiting on a resource held by the first (or an equivalent situation with more threads involved).

To instrument the source code for data race and deadlock detection the code is compiled with a special flag, executed under control of the `collect -r` command, and then loaded into the Thread Analyzer.

- Applications are first compiled with the `-xinstrument=datarace` compiler flag. It is recommended that the `-g` flag also be set, and that no optimization level be used to help ensure that the line numbers and call-stacks information is returned correctly.
- Resulting application code is then executed within the `collect -r` command allowing for the collection of key runtime information. Use the `collect -r all` option to run the program and create a data race detection and deadlock detection experiment during the execution of the process. Alternately, either data races or dead lock conditions for the experiment.

```
% collect -r race <app> <params>
```

```
% collect -r deadlock <app> <params>
```

- Finally, the results of the experiment are loaded into the Thread Analyzer to identify data race and deadlock conditions (Figure 6-4).

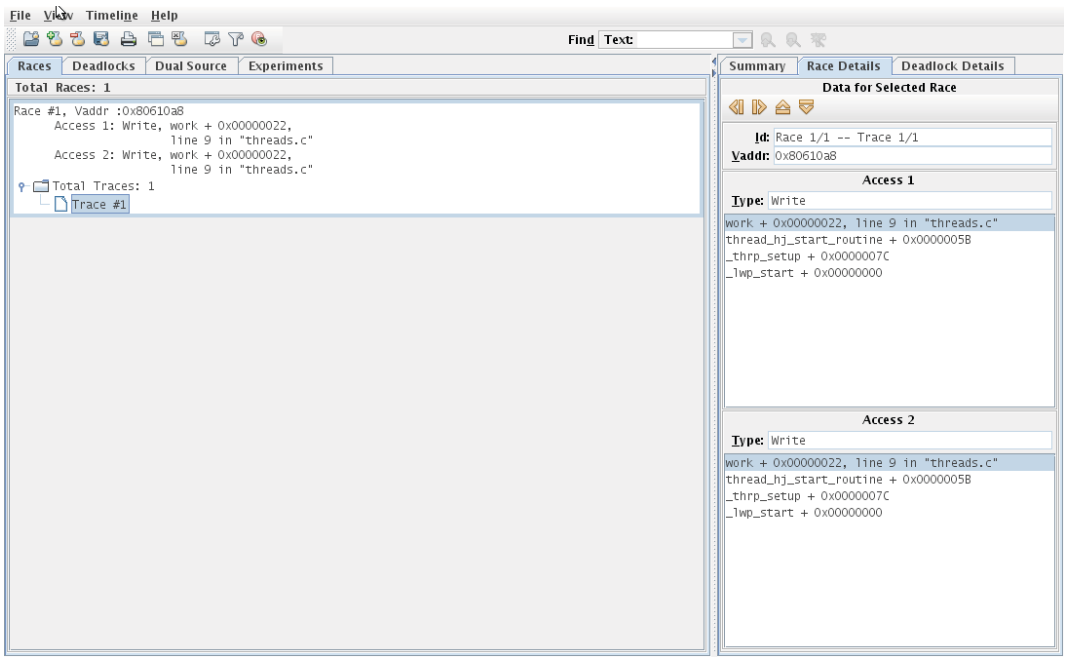


Figure 6-4. Data race conditions can be identified through use of the Thread Analyzer.

The Thread Analyzer can also help identify individual lines of source code that are associated with race conditions (Figure 6-5).

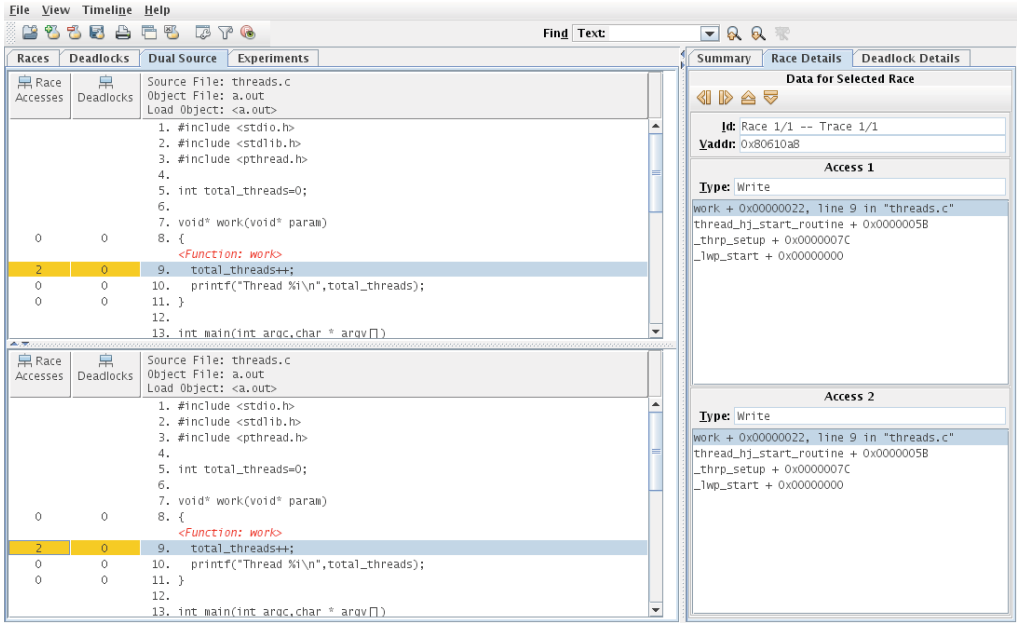


Figure 6-5. Individual lines of source code associated with data race conditions can also be identified using the Thread Analyzer.

Note: See "Optimizing Applications with Oracle Solaris Studio Compilers and Tools" for more information.

Chapter 7 Threads and Multiprocessing

Threading Models

To maximize performance, applications can be architected to execute many tasks simultaneously. To make this possible, the operating system provides support for concurrent processing, such as multiple threads, shared memory, and asynchronous I/O. Multithreaded application programs aim to improve parallelism, and can be written without regard to the number of CPUs configured on the target machine.

Oracle Solaris uses a highly tuned and tested 1:1 thread model in preference to the historic MxN implementation. By simplifying the underlying thread implementation, existing applications can see dramatic performance and stability improvements without requiring recompilation.

While Oracle Solaris supports POSIX and Oracle Solaris threads, the POSIX (`pthread`) model is recommended for new application development and porting efforts. In the POSIX model:

- The kernel uses one thread to handle each system call and interrupt, allowing multiple CPUs to accelerate kernel tasks. Kernel threads are also known as lightweight processes (LWPs).
- Every user level thread has a dedicated kernel thread or LWP (1:1 model). Previous generations of the operating system employed an MxN model. Starting with Oracle Solaris 9, the MxN model was replaced with a 1:1 model based on performance studies and extensive workload analyses of large enterprise customer workloads with very large numbers of processors.

In Oracle Solaris, the original `libthread` (Oracle Solaris Threads) and `libpthread` (POSIX thread) libraries were merged into the standard `libc` library. While it is possible, mixing the threading models creates undue complexity and should be avoided.

Over 100 standards POSIX functions are available through the `pthreads` API. See the `pthreads(5)` man page for detailed information, including a comparison to the Oracle Solaris threading APIs.

Differences Between Oracle Solaris and HP-UX Threading Models

Since HP-UX 11i v3 and Oracle Solaris implement the same POSIX 1003.1c standard, any differences in the interface are largely due to edge cases left unspecified by the standard, or by permitted implementation dependences.

Oracle Solaris has a rich range of process scheduling features, and some of this is reflected in the threading model. (For example, Oracle Solaris `pthreads` have a priority attribute that HP-UX v11.3 `pthreads` lack.) While most applications can do very well with the default scheduling, large ensembles of threads and mission-critical enterprise class multiprocess applications can benefit from careful use of the various process scheduler features. See http://download.oracle.com/docs/cd/E19963-01/html/821-1460/rmfss-2.html for an introduction to the scheduler.

Table 7-1 summarizes the differences in threading attributes, interface differences, and unique extensions available in Oracle Solaris.

TABLE 7-1. THREADING SUMMARY

PTHREAD DEFAULT ATTRIBUTES			
ATTRIBUTE	**HP-UX 11I v3**	**ORACLE SOLARIS 10**	**COMMENT**
`stacksize`	256 KB	Depends on system tunable default_stksize	Default is 3xPAGESIZE for SPARC; 2x PAGESIZE for x86 and 5x PAGESIZE for AMD64 systems. Max can be 32x the default value.
`Priority`	NA	0	
`Inheritsched`	`PTHREAD_INHERIT_SCHED`	`PTHREAD_EXPLICIT_SCHED`	A default change to `INHERIT_SCHED` is possible. Use the following rather than accepting the default: `pthread_attr_setinheritsched()`
`Schedpolicy`	`SCHED_TIMESHARE`	`SCHED_OTHER`	`SCHED_OTHER` is the traditional Oracle Solaris time-sharing (TS) scheduling class.
`Guardsize`	`PAGESIZE`	`PAGESIZE`	Pages are 4KB for HP-UX and typically 8 KB for Oracle Solaris, depending on the hardware platform.

NOTABLE PTHREAD INTERFACE DIFFERENCES			
API	**HP-UX 11I v3**	**ORACLE SOLARIS 10**	**COMMENT**
`pthread_create`	`EAGAIN` for errors	Errors reported through `errno`.	Typical failure is due to `ENOMEM`.
`pthread_join`	When called by more than 1 thread one returns, and the others are undefined.	When called by more than 1 thread, one returns normally and the others return `ESRSH`.	
`pthread_key_create`	`_SC_THREAD_KEYS_MAX=431` `_SC_THREAD_DESTRUCTOR_ITERATIONS=430`	`pthread_key_create`	
`pthread_getschedparam` `pthread_setschedparam`	Priority values represent actual scheduling values without reflecting temporary adjustments		
`sched_yield`	On failure, returns -1 and `errono=ENOSYS`	On failure, returns -1, sets error based on the specific failure.	

HP PTHREAD EXTENSIONS WITH NO ORACLE SOLARIS EQUIVALENTS		
pthread_suspend	pthread_processor_id_np	pthread_mutexattr_getyieldfreq_np
pthread_continue	pthread_mutexattr_getspin_np	pthread_mutexattr_setyieldfreq_np
pthread_resume_np	pthread_mutexattr_setspin_np	pthread_default_stacksize_np
pthread_num_processor_np		

Support for Chip-Multithreading Technology

Unlike traditional single-threaded processors and even most current multicore processors, hardware multithreaded processors, such as Oracle's SPARC T3 processor, allow rapid switching between active threads as other threads stall for memory. The key to this approach—each core is designed to switch between multiple threads on each clock cycle. As a result, the processor's execution pipeline remains active doing useful work, even as memory operations for stalled threads continue in parallel.

Thread-rich applications common in commercial workloads benefit greatly from this model, whether comprised of larger multithreaded applications, or large numbers of smaller single-threaded applications. The number of simultaneous threads that can be accommodated is quite large. Oracle Solaris provides specific features that take advantage of chip multithreading technology. Systems based on Oracle's CMT processors appear as a familiar SMP system to the operating system and the applications it supports.

- **CMT awareness.** Oracle Solaris is aware of the CMT processor hierarchy, enabling the scheduler to effectively balance the load across available pipelines. Even though it exposes the processor as multiple logical processors, the operating system understands the correlation between cores and the threads they support.
- **Fine granularity.** Oracle Solaris can enable or disable individual processors. In the case of CMT processors, this ability extends to enabling or disabling individual cores and logical processors. In addition, standard operating system features, such as processor sets, provide the ability to define a group of logical processors and schedule processes or threads on them.
- **Binding interfaces.** Oracle Solaris allows considerable flexibility. Processes and individual threads can be bound to either a processor or a processor set, if required or desired.

Chapter 8 Distributing Applications

In HP-UX 11i v3 and Oracle Solaris, software is delivered in units called *packages*, collections of files and directories that are required for a software product. Depending on the size of the application, one or more packages may be needed for distribution. Packages contain *package objects,* that are the application files to be installed, and *control files* that control how, where, and if the package is installed.

HP-UX provides tools for packaging applications in the HP-UX Software Distributor format. On Oracle Solaris, developers gather the package objects (application files and directories), required information files (`pkginfo` and `prototype` files), and optional information files and installation scripts, and build the package using the `pkgmk` command.

Preparing an Application for Packaging

Applications can be distributed in one or more packages. While many small packages lengthen installation time, distributing an application as a single, large package is not always possible. If multiple packages are desired or required, care must be taken in how the application code is segmented. Consider the following guidelines (presented in order of importance) when planning and building packages for Oracle Solaris distribution.

- **Make packages remotely installable.** Doing so enables administrators to install the application on remote desktops and servers.
- **Optimize for client-server configurations.** Be sure to consider the software configuration when designing packages. Good packaging design divides the affected files to optimize installation. For example, the contents of the root (`/`) and `/usr` file systems should be segmented so that server configurations can be easily supported.
- **Package on functional boundaries.** Packages should be self-contained and distinctly identified with a set of functionality. For example, a package that contains UFS should contain all UFS utilities and be limited to only UFS binaries.
- **Package on royalty boundaries.** Put code that requires royalty payments in a dedicated package or group of packages. Do not disperse the code into more packages than necessary.
- **Package by system dependencies.** Keep system-dependent binaries and architecture-dependent binaries in dedicated packages. For example, do not mix binaries for SPARC platforms with binaries for x86 systems.
- **Eliminate package overlap.** Eliminate duplicate files whenever possible to minimize support and versioning issues.
- **Package on localization boundaries.** Localization-specific items should be in their own package. An ideal packaging model would have a product's localizations delivered as one package per locale. International defaults can be delivered in a package. This design isolates the files that are necessary for localization changes and standardizes the delivery format of localization packages.

Building a Package

The following steps outline the process for building a package.

- Create a `pkginfo` file that describes the characteristics of the package using a text editor. Define the package name, description, platform architecture it runs on, version number, software category, and base directory for installation. The following example shows the contents of a sample `pkginfo` file.

```
PKG=SUNWcadapp
NAME=CAD application for designing chips. Runs on SPARC hardware and is installed
in the usr partition.
ARCH=sparc
VERSION=release 1.0
CATEGORY=system
BASEDIR=/opt
```

- Organize the package contents into a hierarchical directory structure.
- Create information files that define package dependencies, include a copyright, and reserve space on the target system. (This step is optional.)
- Create installation scripts that customize package installation and removal. (This step is optional.)
- Create a `prototype` file that describes the object in the package. For detailed information see http://download.oracle.com/docs/cd/E19082-01/817-0406/ch2buildpkg-16803/index.html.
- Build the package using the `pkgmk` command.
- Verify and transfer the package to a distribution medium.

More information on designing and building packages can be found in the *Application Packaging Developer's Guide*, located at http://download.oracle.com/docs/cd/E19082-01/817-0406/index.html.

Important Note: A new Image Packaging System (IPS), currently available in Oracle Solaris 11 Express, is set to be the preferred model for packaging applications in Oracle Solaris 11.

Chapter 9 Running Applications

Oracle Solaris Service Management Facility

Operating environments provide, and consist of, many services—including infrastructure services such as file systems, network stacks, logging and security, and application services such as Web servers and databases. Services almost never stand alone. They rely on other services to perform tasks. For example, a Web application service might depend on network and local file system services. Keeping track of these relationships, ensuring that all processes comprising a service are operating, and automating error recovery in the event a service fails can be complex and error prone.

The traditional startup script methods used in HP-UX 11i v3, older Oracle Solaris releases, and other UNIX and similar environments complicate service administration. Oracle Solaris 10 introduces a substantively different facility—the Oracle Solaris Service Management Facility (SMF)—that simplifies management and delivers improved ways to control and manage services. Relationships can be defined between services, and services can be dependent on one another in order to run. With a new set of administrative interfaces, SMF allows services to be easily and consistently configured, enabled, and controlled—all while providing better visibility into errors, automated recovery from failures, and improved debugging capabilities to help resolve service-related problems quickly.

Service Components

SMF services are comprised of one or more components that are utilized based on the service functionality and categorization (Figure 9-1). SMF *service manifests* are the delivery mechanism for services. Created as XML files that are imported into the SMF repository, SMF manifests contain a complete set of properties that are associated with a service or a service instance, including the relationship and dependency information between software services. It also describes the conditions under which failed services can be automatically restarted. SMF uses this information for service management and failure root cause determination. A separate service manifest is required for each service or application.

The service restarter invokes *service methods* to move a service from one state to another whenever an administrative action is performed. These methods are defined in the SMF configuration repository and can be executables, shell scripts, or keywords. In some cases, a *start method* invokes a service executable that provides service capabilities. In addition, SMF managed services use *Fault Management Resource Identifiers* (FMRI) to identify system objects for which advanced fault and resource management capabilities are provided. Log files can be used to capture information about the service as it executes.

Detailed information on SMF, including descriptions of the framework and components, and discussions for using and administering SMF, monitoring services, viewing dependencies, and diagnosing service issues can be found in the "Management of Systems and Services Made Simple with the Oracle Solaris Service Management Facility" white paper located at http://www.oracle.com/technetwork/server-storage/solaris/solaris-smf-wp-167901.pdf.

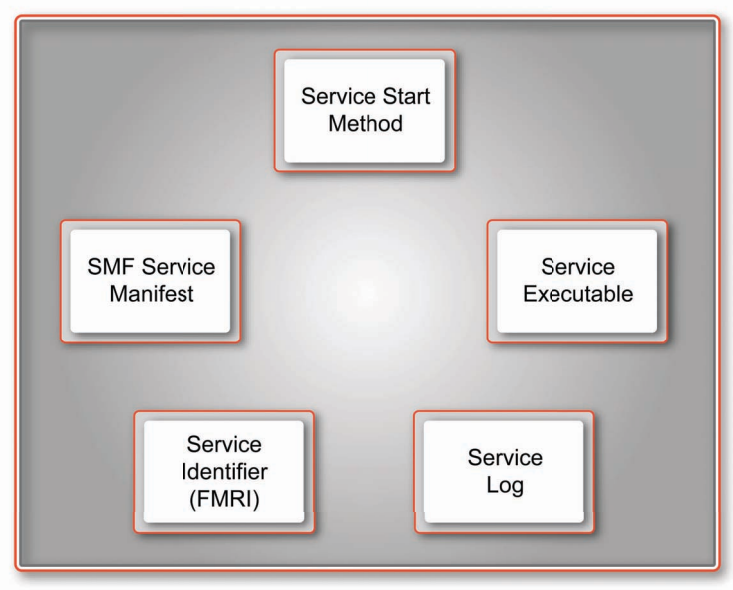


Figure 9-1. SMF consists of several service components.

Creating a Service for an Application

The steps below provide an overview of the process for creating an SMF service for an application.

1. Create a method shell script for the service. The example below uses the service name `foo`.

```
#!/sbin/sh
. /lib/svc/share/smf_include.sh
    if [ -x /opt/SUNWsmftest/bin/foo ]; then
       /opt/SUNWsmftest/bin/foo
    else
       echo "/opt/SUNWsmftest/bin/foo is missing or not executable."
       exit $SMF_EXIT_ERR_CONFIG
    fi

    exit $SMF_EXIT_OK
```

2. Create the XML manifest file. See the "How to Create an Oracle Solaris Service Management Facility Manifest" white paper located at http://www.oracle.com/technetwork/server-storage/solaris/solaris-smf-manifest-wp-167902.pdf for details. The example below shows a sample manifest file for the `foo` service.

```
<?xml version="1.0"?>
<!DOCTYPE service_bundle SYSTEM
"/usr/share/lib/xml/dtd/service_bundle.dtd.1">
<service_bundle type='manifest' name='SUNWcsu:foo'>

<service
    name='system/foo'
    type='service'
    version='1'>

    <single_instance />

    <!-- foo_opt says that foo depends on local filesystem /opt -->
    <dependency
        name='foo_opt'
        type='service'
        grouping='require_all'
        restart_on='none'>
        <service_fmri value='svc:/system/filesystem/local' />
</dependency>

<!-- foo_cron says that foo depends on svc:/system/cron -->
    <dependency
        name='foo_cron'
        type='service'
        grouping='require_all'
        restart_on='refresh'>
        <service_fmri value='svc:/system/cron' />
    </dependency>

    <exec_method
        type='method'
        name='start'
        exec='/opt/SUNWsmftest/lib/svc-foo.sh'
        timeout_seconds='60'>
        <method_context>
        <method_credential user='root' group='root' />
        </method_context>
        </exec_method>

    <exec_method
        type='method'
        name='stop'
        exec=':kill'
```

3. Use the `svccfg` command to read in the XML manifest and add the `foo` service to the SMF repository.

```
# svccfg import /var/svc/manifest/system/foo.xml
```

4. Verify that the `foo` service has been created and defined within SMF.

```
# svcs foo
STATE STIME FMRI
disabled 10:56:21 svc:/system/foo:default_foo
```

5. Enable the service.

```
# svcadm enable foo
```

6. Verify the service is online.

```
# svcs foo
```

7. Verify the daemon is running. The service now is available to handle service requests.

```
# ps -ef | grep foo
root 753 1 89 10:57:11 ? 0:48 /opt/SUNWsmftest/bin/foo
```

Comparison of Administration Commands

Table 9-1 lists the commands used to administer services in other UNIX environments, such as HP-UX and previous Oracle Solaris releases, with the comparable SMF commands.

TABLE 9-1. FREQUENTLY USED COMMANDS FOR ADMINISTERING SMF SERVICES

TASK	OTHER UNIX PROCEDURE	SMF PROCEDURE
Disable a system service (Ex: `cron`)	`rm /etc/rc2.d/S75cron` (Repeat after every `cron` patch application and system upgrade.)	`svcadm disable cron`
Re-enable a service	Reinstall `/etc/rc2.d/S75cron`.	`svcadm enable cron`
Enable `inetd` services (Ex: `finger`)	Edit the `/etc/inet/inetd.conf` file. Uncomment the service to be enabled and save. Issue this command: `pkill -HUP inetd`	`svcadm enable finger`
Stop services	`/etc/init.d/sshd stop`	`svcadm disable –t ssh` Disable lasts until reboot.
Start services	`/etc/init.d/sshd start`	`svcadm enable -t ssh`
Restart services	`/etc/init.d/sshd stop` `/etc/init.d/sshd start`	`svcadm restart ssh`
Refresh configuration	`kill -HUP `cat /var/run/sshd.pid``	`svcadm restart ssh`

Continued Support for .rc Scripts

While many standard Oracle Solaris 10 services are now managed by the Oracle Solaris Service Management Facility, scripts placed in `/etc/rc*.d` continue to execute on run-level transitions. Most of the `/etc/rc*.d` scripts included in previous Oracle Solaris releases have been removed as part of the transition to SMF. Because Oracle Solaris 10 retains the ability to run the remaining scripts, third-party applications can be deployed without the need for conversion to SMF.

The `/etc/inittab` and `/etc/inetd.conf` files remain available for packages to amend with post-install scripts. These legacy-run services can be added to the service configuration repository via the `inetconv` command. While service status can be viewed through SMF, other changes are not supported. Applications that use this feature do not benefit from the precise fault containment provided by SMF.

Applications that convert to using SMF should not modify the `/etc/inittab` and `/etc/inetd.conf` files. Converted applications do not use `/etc/rc*.d` scripts, and the new version of the `inetd` daemon does not look for entries in the `/etc/inetd.conf` file.

Chapter 10 File Systems and Data

When porting applications from HP-UX 11i v3 to Oracle Solaris, it is important to understand whether and how file systems and data are potentially affected, how best to move data from one platform to another, and which file systems offer new capabilities.

File Systems

The HP-UX 11i offers a rich suite of file systems. Oracle Solaris supports many of these, enabling users to simply mount existing file systems rather than migrate them. Table 10-1 lists common file systems and details their availability on HP-UX 11i and Oracle Solaris.

TABLE 10-1. SUPPORTED FILE SYSTEMS

FILE SYSTEM	DESCRIPTION	HP-UX 11i v3	ORACLE SOLARIS 10
CacheFS	Used to improve the performance of remote file systems or slow devices	√	√
CDFS	CD-ROM file system	√	√
CTFS	Contract file system, used to create, control, and observe contracts (primarily used by SMF)	—	√
FDFS	File Descriptor File Systems, provides explicit names for opening files using file descriptors	√	√
FIFOFS	First-in, first out file system, provides named pipe files that give processes common access to data	√	√
HSFS	High Sierra File System, ISO 9660, the first CD-ROM file system	√	√
LOFS	Loopback file system, allows the creation of a virtual file system so that files can be accessed using an alternative path name	√	√
MemFS	Memory File System	√	—
MNTFS	Provides read-only access to the table of mounted file systems for the local system	√	√
NAMEFS	Used mostly by STREAMS for dynamic mounts of file descriptors on top of files	√	√
NFS	Network File System	√	√

FILE SYSTEM	DESCRIPTION	HP-UX 11I v3	ORACLE SOLARIS 10
OBJFS	Object file system, describes the state of modules currently loaded by the kernel (used by debuggers to access information about kernel symbols without having to access the kernel directly)	—	√
PCFS	Supports read and write access to data and programs on DOS-formatted disks	√	√
Oracle Solaris ZFS	A general-purpose, enterprise-class file system that integrates traditional file system functionality with built-in volume management techniques and data services	—	√
Oracle's Sun QFS	Provides nearly raw device access to information and data consolidation for read/write file sharing	—	√
Oracle's Sun SAM-FS	Provides data classification, centralized metadata management, policy-based data placement, and migration	—	√
SHAREFS	Provides read-only access to the table of shared file systems for the local system	—	√
SPECFS	Special file system, provides access to character special devices and block devices	√	√
SWAPFS	Used by the kernel for swapping	√	√
TMPFS	Uses local memory for file system reads and writes, which is typically faster than a UFS file system	√	√
UDFS	Universal Disk Format file system, the industry-standard format for storing information on optical media such as DVDs	√	√
UFS	UNIX file system	√	√

UFS

The popular UNIX file system (UFS) remains the default file system in Oracle Solaris 10. While 32-bit and 64-bit HP-UX environments running UFS are limited to 1 TB file systems, Oracle Solaris 10 provides support for multi-terabyte UFS file systems when running a 64-bit Oracle Solaris 10 kernel. All UFS file systems and utilities are updated to support multi-terabyte UFS file systems.

Oracle Solaris ZFS

Developers often are forced to divide large datasets into segments that fit within file system size limits, a process that can be difficult and error prone. Oracle Solaris ZFS provides virtually unlimited file system scalability to large-scale applications. File systems can grow to 21 billion Yottabytes, enabling developers

to place extremely large datasets in a single file system on a pool of storage and optimize it for performance. In addition, Oracle Solaris ZFS frees developers from worrying about, and coding for, some classes of data integrity errors. On-disk data is kept self-consistent and silent data corruption is eliminated.

Integrated Volume Management and Storage Pools

Unlike traditional file systems that require a separate volume manager, Oracle Solaris ZFS integrates volume management functions. Breaking free of the typical one-to-one mapping between the file system and its associated volumes, Oracle Solaris ZFS introduces the storage pool model (Figure 10-1). It decouples the file system from physical storage in the same way that virtual memory abstracts the address space from physical memory, allowing for more efficient use of storage devices. Space is shared dynamically between multiple file systems from a single storage pool, and is parceled out of the pool as file systems request it. Physical storage can be added to storage pools dynamically, without interrupting services, providing new levels of flexibility, availability, scalability, and performance. When capacity no longer is required by a file system in the pool, it is made available to other file systems.

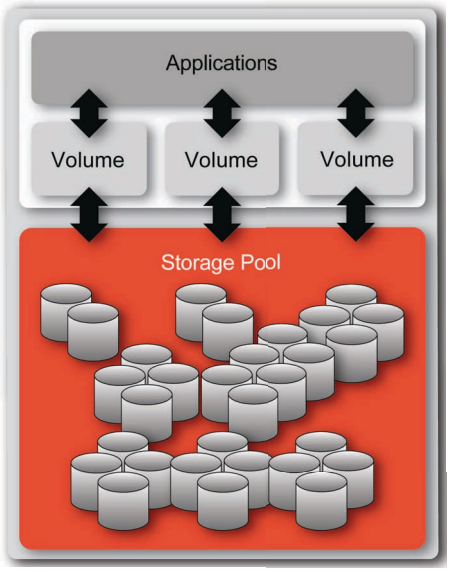


Figure 10-1. Virtual storage pools let multiple file systems share storage space.

Hybrid Storage Pools

Oracle Solaris ZFS gives developers the ability to optimize data placement for fast access. Rather than augmenting the storage infrastructure with expensive disks, Flash technology can be placed in a new storage tier to assist hard disk drives by holding frequently accessed data to minimize the impact of disk latencies and improve application performance. By using Flash devices to handle certain types of I/O, and hard disk drives to store massive data sets, a Hybrid Storage Pool delivers significant application performance gains without sacrificing capacity (Figure 10-2).

Hybrid Storage Pool technology is designed to exceed the performance of Fibre Channel technologies without the additional management complexity of a SAN. Several Oracle Solaris ZFS components are key to Hybrid Storage Pool operation and help accelerate application performance.

- The Oracle Solaris ZFS Adaptive Replacement Cache (ARC) is the main file system memory cache and is stored in DRAM.
- The Level Two Adaptive Replacement Cache (L2ARC) extends the ARC into read-optimized Flash devices to provide a large read cache to accelerate reads. The Oracle Solaris ZFS Intent Log (ZIL) is transactional and uses write-based Flash devices to provide a large cache to accelerate writes.
- The disk storage pool consists of conventional disk drives. Note that high-performance, expensive disk drives are no longer strictly required to achieve high performance levels given the interposition of Flash devices in a Hybrid Storage Pool.

Sophisticated file system algorithms in Oracle Solaris ZFS use the ARC in memory and the L2ARC on Flash devices to determine pre-fetch or data placement during sustained read operations. Flash devices accelerate write throughput for Oracle Solaris ZFS synchronous write I/O operations, helping to boost write performance.

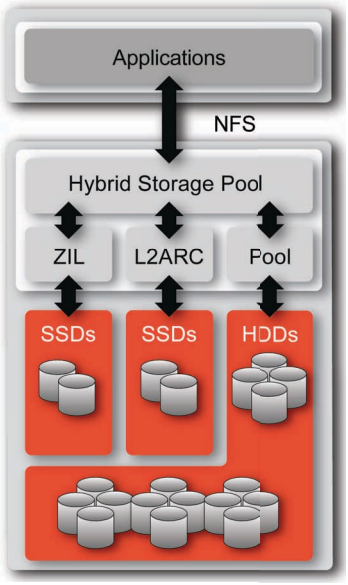


Figure 10-2. Hybrid Storage Pools optimize data placement to improve I/O performance.

Data Integrity

Rather than build complex data integrity checks into applications, developers can depend techniques such as copy-on-write and end-to-end checksumming built into Oracle Solaris ZFS to keep on-disk data self-consistent and eliminate silent data corruption. Data is written to a new block on the media before changing the pointers to the data and committing the write. Because the file system is always consistent,

time-consuming recovery procedures like `fsck` are not required if the system is shut down in an unclean manner. In addition, data is read and checked constantly to help ensure correctness, and any errors detected in a mirrored pool are automatically repaired to protect against costly and time-consuming data loss and (previously undetectable) silent data corruption. Corrections are made possible by a RAID-Z implementation that uses parity, striping, and atomic operations to help reconstruct corrupted data.

File System Size

The maximum file size supported by a file system can limit applications. When the maximum file system size is reached, workarounds often are required, such as splitting a file or database into multiple parts or distributing it across multiple machines. The standard HP-UX file system scales to 1 TB. On Oracle Solaris 10, UFS scales to 16 TB, and combined with the Sun SAM-FS file system and Sun QFS software, scales to 252 TB. The 128-bit Oracle Solaris ZFS scales to 16 exabytes (EB). As a result, programmers migrating data from HP-UX to Oracle Solaris should not experience file system limitations that hamper application development or testing efforts.

Data Transformation

Data transformation is the process of converting data from one format to another, and is an important component of any porting effort if data is to be readable on the target system. Data transformation can involve file systems, file content, applications, and database content.

Encoded Data Transformations

Encoded data transformations are necessary when data is stored in a different or incompatible file format than the receiving system anticipates. Fortunately, HP-UX and Oracle Solaris both use ASCII to store textual data, as well as a standard text file format. As a result, issues stemming from the use of other character sets, such as EBCDIC, and differences in text file formatting, such as the use of control-linefeed (CR/LF) versus carriage return (CR) characters to delimit lines in a file, are avoided.

Application Data Transformation

HP-UX and Oracle Solaris provide many common applications and utilities for managing data. For example, the tape archive utility (`tar`) uses a similar data format and provides many common options in both environments. As a result, developers already familiar with the `tar` utility in the HP-UX environment are able to be immediately productive on Oracle Solaris. This commonality is true for many other applications and utilities, and can yield significant benefits both during and after the data migration. For those applications that differ between HP-UX, and Oracle Solaris, most provide a utility to convert standard data interchange formats, such as comma separated values or tab delimited files, into their format.

Database Transformation

Many enterprise applications depend on large databases. If an older version of a database is in use in the HP-UX environment, licenses may or may not be available for those versions on Oracle Solaris. Developers should be prepared to acquire a current version of the database software. It is important to note that changes to the infrastructure may be needed to support the new database and its configuration, however existing data should be immediately accessible. Table 10-2 lists the popular databases supported on HP-UX 11i v3 and Oracle Solaris.

TABLE 10-2. SUPPORTED DATABASES

DATABASE	HP-UX 11I v3		ORACLE SOLARIS 10	
	PA-RISC	ITANIUM	SPARC	X86
Oracle Database 11*g* Release 2	√	√	√	√
Oracle Database 11*g* Release 1	√	√	√	√
Oracle Database 10*g* Release 2	√	√	√	√
MySQL Database 5.6	√	√	√	√
MySQL Database 5.5	√	√	√	√
MySQL Database 5.1	√	√	√	√
MySQL Database 5.0	√	√	√	√
Sybase IQ Enterprise Edition 15.3	√	√	√	√
Sybase IQ Enterprise Edition 15.2	√	√	√	√
Sybase IQ Enterprise Edition 15.1	√	√	√	√
PostgreSQL Database	√	√	√	√

While there are many similarities between a database running on HP-UX and one running on Oracle Solaris, simply moving a database from one to the other likely requires some data transformation. In the case of the same database vendor in both environments, this may be as simple as exporting the database running on HP-UX to a standardized file format, followed by an import into a new database on Oracle Solaris. When the port also involves a change in database vendors, more extensive data transformations may be required.

Because database transformations are usually such a large part of the overall porting effort, many specialized utilities have been created to address them. These programs, called Extract, Transform, and Load (ETL) utilities, take a wide array of formats and convert them into Structured Query Language (SQL) for relational database management systems (RDBMS). Most RDBMSs provide a basic set of utilities to convert SQL or standard interchange formats into their data storage format.

Chapter 11 Clustering

As datacenters transition from HP-UX 11i v3 running on HP servers to Oracle Solaris running on Oracle servers, developers working on highly available applications and services need to replace HP Serviceguard with a different clustering technology. Oracle offers Oracle Real Application Clusters and Oracle Solaris Cluster software.

Oracle Real Applications Cluster

Database environments can take advantage of Oracle Real Application Clusters (RAC), a cluster database with a shared cache architecture that overcomes the limitations of traditional shared-nothing and shared-disk approaches. By supporting the transparent deployment of a single database across a server pool, Oracle RAC provides fault tolerance from hardware failures or planned outages. Server pools can be scaled as needed, with up to 100 servers supported in a pool when combined with Oracle Clusterware. Using a High Availability API, developers can integrate applications into a platform that can monitor, relocate, and restart database applications when needed.

Oracle Solaris Cluster

Integrated with Oracle Solaris, Oracle Solaris Cluster provides load balancing, automatic fault detection, and failover to keep mission-critical applications and services in traditional or virtualized environments highly available. Migrating an application from the HP Serviceguard environment to the Oracle Solaris Cluster environment involves creating an agent that specifies the actions to be taken should the application fail. Application source code does not need to be modified. By migrating from HP Serviceguard to Oracle Solaris Cluster, businesses continue to have a scalable and flexible solution that is suited equally to a small local cluster or larger extended clusters that can be a part of the enterprise disaster recovery strategy.

Overview

At its simplest, Oracle Solaris Cluster monitors the health of cluster components, including the stack of applications, middleware, operating system, servers, storage, and network interconnects. Any failure executes a policy-based, application-specific recovery action. Recovery is enabled through redundant infrastructure and intelligent software algorithms.

From a physical perspective, an Oracle Solaris Cluster system consists of two or more servers that work together as a single entity to cooperatively provide applications, system resources, and data to users (Figure 11-1). Each server provides some level of redundancy. Data is stored on highly available redundant disk systems, which may be mirrored, supporting data access in the event of a service interruption on a single disk or storage subsystem. Redundant connections are provided to the disk systems so that data is not isolated in the event of a server, controller, or cable failure. A high-speed, redundant, private interconnect provides access to resources across the server set. Redundant connections to the public network also provide each node with multiple paths for access to outside systems, helping ensure continued access in the event of a network connection or node failure.

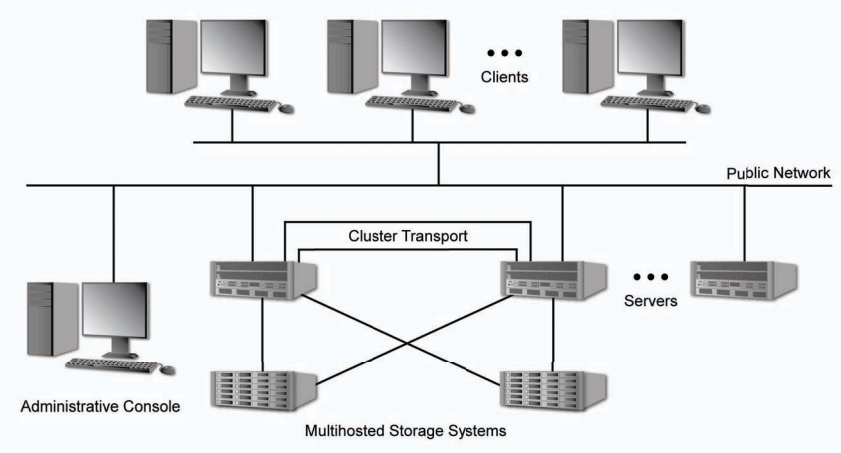


Figure 11-1. Oracle Solaris Cluster enables multiple servers and storage systems to act as a single system.

No single hardware, software, storage, or network failure can cause the cluster to fail. Loss of service is prevented through hardware redundancy, hardware and software failure detection, automatic recovery of services, and application failover. In addition, a single management view enables the entire cluster to be managed as a single entity, reducing the risk of errors.

Oracle Solaris Cluster includes capabilities to detect, isolate, and contain failing cluster nodes. It accomplishes this using a robust, kernel-based membership monitor. Each node in the cluster sends out low-level data link provider interface (DLPI) packets once per second (a *heartbeat*) to each of its peers on each of the private networks. These packets are sent in the kernel interrupt context, making them very resilient to peaks in system load. A network, or path, between two nodes is declared down only if a heartbeat message does not complete the round trip between nodes, over that specific path, within the timeout period.

Network Availability

Oracle Solaris Cluster leverages Oracle Solaris IP network multipathing (IPMP) as public network interfaces for monitoring local failures, and for performing automatic failover from one failed network adaptor to another. IP network multipathing enables a server to have multiple network ports connected to the same subnet. First, IP network multipathing software provides resilience from network adapter failure by detecting the failure or repair of a network adapter. The software simultaneously switches the network address to and from the alternate adapter. When more than one network adapter is functional, IP network multipathing increases data throughput by spreading outbound packets across multiple adapters.

For scalable data services, requests go through a round-robin load-balancing scheme for a balanced load distribution to the various instances of the distributed application running within the cluster. Scalable data services can be made more secure through the use of IPsec services in combination with Oracle Solaris Cluster load balancing services.

Data Integrity

Because cluster nodes share data and resources, Oracle Solaris Cluster works to ensure a cluster never splits into separate, active partitions that continue to access and modify data. Similar to HP Serviceguard, Oracle Solaris Cluster applies *fencing* techniques and a *quorum* to protect data integrity. Failing nodes are isolated from the cluster and prevented from accessing clustered data. The fencing protocol can be chosen per storage device.

In a more complex situation where all paths across the private interconnect fail and the cluster breaks into multiple partitions, Oracle Solaris Cluster uses a quorum mechanism to recreate the cluster and resolve partitions or *split brain syndrome*, and to protect data integrity. The quorum also prevents *amnesia* by detecting and rejecting the use of outdated configuration information that could lead to data corruption. The quorum can be tailored to the storage and system topology, enabling disk-based and software quorum solutions. A quorum device protocol permits the use of different types of disks, such as high-capacity 2 TB disk drives, SATA, and Flash as quorum devices. All quorum devices are continuously monitored to enhance availability.

Key Components

Key components of Oracle Solaris Cluster include:

- **High availability framework.** The framework detects node failures quickly and activates resources on another node in the cluster. It includes a Cluster Membership Monitor, a distributed set of algorithms and agents that exchange messages over the cluster interconnect to enforce a consistent membership view, synchronize reconfiguration, handle cluster partitioning, and help maintain full connectivity among all cluster members. Inter-node message delivery and responses are handled in an atomic manner that accounts for delivery failures, node membership, and software revision level (to provide for rolling upgrades).
- **Failover, scalable, and cluster-aware agents.** Failover and scalable agents are software programs that support Oracle or third-party applications to take full advantage of Oracle Solaris Cluster features. Cluster-aware applications have direct knowledge of Oracle Solaris Cluster systems, such as Oracle Real Application Clusters (RAC) software.
- **Highly available private interconnect.** Multiple types of interconnect technologies are supported by Oracle Solaris Cluster to establish a private communication channel between cluster nodes. Support for multiple interconnects helps ensure high availability and improve performance of private inter-node communication. *Heartbeats* monitor cluster nodes over the private interconnect. If a server goes offline and ceases its heartbeat, it is isolated. Applications and data are failed over to another server quickly and transparently to users.

Key Features

Oracle Solaris Cluster extends Oracle Solaris to provide enhanced availability of hosted applications. Using the advanced capabilities in Oracle Solaris, Oracle Solaris Cluster offers:

- **Flexible configurations.** While HP Serviceguard supports N+1 clusters, Oracle Solaris Cluster supports pair, pair+N, N*1, N*N for flexible topologies.
- **Global devices, files, and networking.** All global devices, files, and network interfaces can be seen as local resources. Cluster nodes can access and utilize devices that are attached to another node within the cluster. These facilities create improved resource availability and simplified administration.
- **Virtualization support.** Oracle Solaris Cluster supports Oracle's virtualization portfolio—Oracle Solaris Containers, Oracle VM Server for SPARC, and Dynamic Domains (available on Oracle's SPARC Enterprise M-Series servers)— for flexible configurations that support consolidation efforts. Applications can run unmodified in a virtualized environment.
- **Flexible storage support.** Oracle Solaris Cluster deployments can take advantage of a wide range of storage technologies, such as Fibre Channel, SCSI, iSCSI, and NAS storage solutions from Oracle and other vendors. Support for a broad range of file systems, including Oracle Solaris ZFS, and volume managers eases the data migration process.
- **Oracle RAC 10*g* and 11*g* integration and administration.** Automated installation and wizard-led configuration enable faster setup of Oracle RAC with Oracle Solaris Cluster. Specific Oracle RAC integration points enable improved coordination and simplified administration.
- **Campus and geographic clusters.** Oracle Solaris Cluster supports the creation of clusters across a campus or metropolitan area (campus cluster) or over large distances (geographic cluster) to support multi-site disaster recovery.

Writing an Agent for Oracle Solaris Cluster

While HP Serviceguard requires developers to write scripts for applications, Oracle Solaris Cluster includes an Agent Builder tool that automates the creation of a data service. Developers supply Agent Builder with information about the application and data service to be created, such as whether a scalable or failover agent is desired, whether the service is network-aware, the commands to use to start and stop the application, etc. Agent Builder generates the data service, including source and executable code (C or Korn shell), a customized Resource Type Registration (RTR) file, and an Oracle Solaris package for distribution.

Agent Builder can be used to generate a complete data service for most enterprise applications, with only minor changes on the part of the developer. Developers creating applications with more sophisticated requirements, such those needing validation checks for additional properties, can use Agent Builder to generate most of the code and manually make additions. At a minimum, Agent Builder can be used to generate the final Oracle Solaris installation package. More information on using Agent Builder can be found in the *Oracle Solaris Cluster Data Services Developer's Guide*.

Differences Between HP Serviceguard and Oracle Solaris Cluster

Table 11-1 summarizes the key differences between HP Serviceguard and Oracle Solaris Cluster.

TABLE 11-1. COMPARISON SUMMARY OF HP SERVICEGUARD AND ORACLE SOLARIS CLUSTER

ITEM	HP SERVICEGUARD	ORACLE SOLARIS CLUSTER
Configuration	• 2 to 16 nodes • Active/active, active/standby, rolling standby • N+1	• 2 to 16 nodes (SPARC), 2 to 8 (x86) • Active/active, active/standby, rolling standby • Pair, pair+N, N*1, N*N
Interconnects	• Ethernet, Fast Ethernet, Gigabit Ethernet • FDDI, Token Ring, HyperFabric2, Serial	• Ethernet, Fast Ethernet, Gigabit Ethernet • 10 Gigabit Ethernet, InfiniBand
Networking Protocols	• IPv4, IPv6, RDS	• IPMP, Trunking, Jumbo Frames, VLAN • IPv4, IPv6, SCTP, RDS
Disk Fencing	• Only when using VxFS	• Yes
File Systems	• Veritas VxFS	• Root: UFS, ZFS, Veritas VxFS • Failover: UFS, ZFS, QFS, NFS, VxFS • Cluster: PxFS, Oracle Automatic Storage Management Cluster FIle System (ACFS), QFS
Volume Management	• Veritas Volume Manager • HP-UX Logical Volume Manager	• Oracle Solaris Volume Manager • Veritas Volume Manager • Oracle Automatic Storage Management • Oracle Solaris ZFS
Virtualization Support	• vPars	• Oracle Solaris Containers • Oracle VM Server • Dynamic Domains (on supported systems)
Monitoring	• System (heartbeat) • Network • Application	• System (heartbeat) • Network • Application • Quorum • Disk path • Storage resources
Workload Management	• Yes	• Yes
Cluster Management	• HP Event Monitoring Service • HP Serviceguard Manager	• Web-based GUI • Configuration Wizards • Object-oriented command line interface • Integrated with Oracle Enterprise Manager Ops Center • Integrated with SMF

Agents	• HP Serviceguard Extension for RAC • HP Serviceguard Extension for SAP R/3 • IBM DB2 • Informix • NFS • Oracle Database • Oracle RAC • Sybase	• Oracle Application Server • Oracle Business Intelligence Enterprise Edition • Oracle Communications Calendar Server • Oracle Communications Instant Messaging Server • Oracle Communications Messaging Exchange Server • Oracle E-Business Suite • Oracle Grid Engine Sun Service Provisioning System • Oracle iPlanet Web Server • Oracle iPlanet Web Proxy Server • Oracle Solaris Containers (HA Agent) • Oracle VM Server for SPARC (HA Agent) • Oracle Database • Oracle Grid Engine • Oracle RAC 10*g* and 11*g* • Oracle WebLogic Server • Oracle Business Intelligence Enterprise Edition • Oracle TimesTen • Oracle’s PeopleSoft Enterprise • Oracle’s Siebel CRM • Agfa IMPAX • Apache Proxy Server (HA and scalable) • Apache Web Server (HA and scalable) • Apache Tomcat • DNS • DHCP • IBM WebSphere MQ • IBM WebSphere Message Broker • Informix Dynamic Server • Kerberos • MySQL, MySQL Cluster • NFS • PostgreSQL • Samba • SAP, SAP liveCache, SAP Enqueue Server • SAP SAPDB/Max DB • SWIFT Alliance Access, SWIFT Alliance Gateway • Sybase ASE • IBM DB2 (available from third-party) • Symantec Netbackup (available from third-party)

Chapter 12 Building Secure Applications

Oracle Solaris provides a sophisticated network-wide security system that controls the way users access files, protect system databases, and use system resources. From integrated security services and applications, to enhanced encryption algorithms, to an enterprise firewall for network protection, Oracle Solaris sets a high standard for operating system security by addressing security needs at every layer. Extended security features are also available, including authentication, data integrity, data privacy, and single sign-on capabilities so that tampering, snooping, and eavesdropping do not compromise data or associated transactions.

- **Harden the system**. Oracle Solaris provides security features previously only found in Oracle's Trusted Solaris OS, delivering a secure environment right out of the box. The system can be further hardened and minimized, helping to reduce the risk that a system or application can be compromised. Reduced configurations can be installed—with fewer software packages, no active networking, a minimum number of running services, and enhanced security. Such configurations reduce install time and provide a secured building block for customized deployments.
- **Reduce security risks**. Oracle Solaris offers Role-Based Access Control and Process Rights Management. These technologies reduce security risk by granting users and applications only the minimum capabilities needed to perform tasks. Discrete privileges can be granted—or denied—to any process on the system to create effective security policies, minimize the likelihood of hostile actions, control access to data, and ensure compliance with regulatory requirements.
- **Improve data security policies**. An optional layer of secure label technology in Oracle Solaris, Oracle Solaris Trusted Extensions, allows data security policies to be separated from data ownership. With the ability to support multilevel data access policies, the operating system can help companies meet strict government regulatory compliance goals without modifying existing applications for underlying hardware platforms. It provides a platform for deploying high security desktops, database servers, firewalls, and communication gateways, as well as any enterprise application where access to sensitive information or networks must be strictly controlled.
- **Take advantage of on-board cryptography**. Provided in Oracle servers with UltraSPARC T1, T2, T2+ or SPARC T3 processors, on-chip cryptographic acceleration eliminates the need for additional coprocessor cards, special licensing, network appliances, or power hungry add-on components. The cryptographic capabilities of these processors can be accessed via a built-in Oracle Solaris Cryptographic Framework that provides kernel-level and user-level consumers access to software-based or hardware-based cryptographic capabilities.

Security Interfaces for Developers

Oracle Solaris provides standardized protocols and interfaces that enable enterprise developers to write applications, libraries, and kernel modules that can take advantage of security technologies. In this model, an application, library, or kernel module that uses security services is called a *consumer*. An application that provides security services to consumers is a *provider* and a *plug-in*. Oracle Solaris provides several public security interfaces.

Privileges and Authentication

Privileged applications can override system controls and check for specific user IDs, group IDs, authorizations, or privileges. Two elements enable fine-grained delegation. A *privilege* is a discrete right that can be granted to an application. With the right privilege, applications can perform operations that would otherwise be prohibited by the operating system. Privileges are enforced at the kernel level. An *authorization* grants permission for performing a class of actions that are otherwise prohibited by a security policy. Authorizations are enforced at the user level.

To use the privilege programming interfaces, include the `priv.h` header file. Appendix D lists the interfaces available for using privileges.

Best Practices for Developing Privileged Applications

The following suggestions can aid the development of privileged enterprise applications.

- **Use an isolated system.** Debugging privileged applications on a production system can compromise security if the application is incomplete.
- **Set IDs properly.** Calling processes must have the proc_setid privilege in its effective set to change its user ID, group ID, or supplemental group ID.
- **Use privilege bracketing.** When an application uses privileges, system security policy is overridden. Privileged tasks should be bracketed and carefully controlled to ensure sensitive information is not compromised.
- **Start with basic privileges.** Privileged applications should start with the basic set of privileges needed for minimal operation and add and subtract privileges as needed.
- **Avoid shell escapes.** The new process in a shell escape can use any of the privileges in the parent process's inheritable set. As a result, an end user can potentially violate trust through a shell escape.

Steps for Developing Applications with Authorizations

Authorizations are stored in the `/etc/security/auth_attr` file. To create an application that uses authorizations, take the following steps:

- Scan the `/etc/security/auth_attr` file for one or more appropriate authorizations.
- Check for the required authorization at the beginning of the program using the `chkauthattr(3SECDB)` function.
- Let the administrator know which authorizations are required for this application, if access is denied in the previous step.

Pluggable Authentication Modules

Pluggable authentication modules (PAM) provide system entry applications with authentication and related security services for managing accounts, sessions, and passwords. Applications such as `login`, `rlogin`, and `telnet` are typical consumers of PAM services. The framework provides a uniform way for

authentication-related activities to take place. This approach enables application developers to use PAM services without having to know the semantics of the policy. Algorithms are centrally supplied, and can be modified independently of individual applications.

The PAM library, `libpam(3LIB)`, is the central element in the PAM architecture. It exports an API, `pam(3PAM)`, that applications can call for authentication, account management, credential establishment, session management, and password changes. The `libpam` library imports a master configuration file, `pam.conf(4)`, that specifies the PAM module requirements for each available service. It also imports a Service Provider Interface (SPI), `pam_sm(3PAM)`, which is exported by the service modules.

Generic Security Service Application Programming Interface

The Generic Security Service Application Programming Interface (GSS-API) enables applications to protect data to be sent to peer applications. It provides secure communications between peer applications, as well as authentication, integrity, and confidentiality protection services, and typically is used in the development of secure application protocols. With GSS-API, developers can write applications generically with respect to security. Security implementations do not have to be tailored to a particular platform, security mechanism, type of protection, or transport protocol, and the details of protecting network data are avoided.

The GSS-API creates a security *context* in which data can be passed between applications—a state of trust between two applications. Applications that share a context recognize each other and can permit data transfers while the context lasts. One or more types of protection, known as *security services*, are applied to the data to be transmitted. Addition GSS-API tasks include data conversion, error checking, delegation of user privileges, information display, and identity comparison.

Simple Authentication and Security Layer

Designed for high-level, network-based applications that use dynamic negotiation of security mechanisms to protect sessions, the Simple Authentication and Security Layer (SASL) framework provides authentication services and optional integrity and confidentiality services to connection-based protocols. Developers of enterprise applications and shared libraries can take advantage of SASL mechanisms for authentication, data integrity checking, and encryption through a generic API.

SASL is particularly appropriate for applications that use the Internet Access Message Protocol (IMAP), Simple Mail Transport Protocol (SMTP), Application Configuration Access Protocol (ACAP), and Lightweight Directory Access Protocol (LDAP), as these all support SASL.

The SASL library, `libsasl`, is a framework that allows properly written SASL consumer applications to use SASL plug-ins that are available on the system. A service provider interface (SPI) is provided for plug-ins to supply services to the `libsasl` library. Applications communicate with `libsasl` through an API. The library can request additional information through callbacks registered by the application. For example, the `libsasl` library can use callbacks to get information from the application to complete authentication. Consumer applications can use callbacks to change search paths for configuration data, verify data, and change default behaviors. In addition, servers can use callbacks to change authorization

policies, supply different password verification methods, and obtain changed password information. Table 12-1 lists the callbacks available to client and server applications.

TABLE 12-1. SASL CALLBACKS

CALLBACK	DESCRIPTION
CALLBACKS AVAILABLE TO CLIENT AND SERVER APPLICATIONS	
SASL_CB_GETOPT	Get a SASAL option.
SASL_CB_LOG	Set the logging function for the `libsasl` library and its plug-ins. The default behavior is to use `syslog`.
SASL_CB_GETPATH	Get a colon-separated list of SASL plug-in search paths.
SASL_CB_GETCONF	Get the path to the SASL server's configuration directory. The default path is `/etc/sasl`.
SASL_CB_LANGUAGE	Specifies a comma-separated list of RFC 1766 language codes, in order of preference, for client and server error messages and client prompts. The default is `i-default`.
SASL_CB_VERIFYFILE	Verifies the configuration file and plug-in files.
CALLBACKS AVAILABLE ONLY TO CLIENT APPLICATIONS	
SASL_CB_USER	Get the client user name (authorization ID). The default is the `LOGNAME` environment variable.
SASL_CB_AUTHNAME	Get the client authentication name.
SASL_CB_PASS	Get a client passphrase-based secret.
SASL_CB_ECHOPROMPT	Get the result for a given challenge prompt. Input from the client can be echoed.
SASL_CB_NOECHOPROMPT	Get the result for a given challenge prompt. Input from the client should not be echoed.
SASL_CB_GETREALM	Set the realm to be used for authentication.
CALLBACKS AVAILABLE ONLY TO SERVER APPLICATIONS	
SASL_CB_PROXY_POLICY	Check that an authenticated user is authorized to act on behalf of a specified user.
SASL_CB_SERVER_USERDB_CHECKPASS	Verifies a plain text password against the caller-supplied user database.
SASL_CB_SERVER_USERDB_SETPASS	Stores a plain text password in the user database.
SASL_CB_CANON_USER	Call an application-supplied user canonicalization function.

Oracle Solaris Cryptographic Framework

The Oracle Solaris Cryptographic Framework provides a set of cryptographic services for kernel-level and user-level consumers. Based on the PKCS#11 public key cryptography standard created by RSA Security, Inc., the framework provides a mechanism and API whereby both kernel- and user-based cryptographic functions can transparently use software encryption modules and hardware accelerators configured on the system. The framework provides various services, including message encryption and message digest, message authentication, and digital signing. It also includes APIs for accessing cryptographic services, and SPIs for providing cryptographic services.

- **`ibpkcs11.so` library.** The framework provides access through the RSA Security Inc. PKCS#11 Cryptographic Token Interface (Cryptoki). Applications must link to the `libpkcs11.so` library.
- **pkcs11_softtoken.so shared object.** This private shared object contains user-level cryptographic mechanisms provided by Oracle.
- **pkcs11_kernel.so shared object.** This private shared object is used to access kernel-level cryptographic mechanisms. It offers a PKCS#11 user interface for cryptographic services that are plugged into the kernel's service provider interface.
- **Pluggable interface.** The pluggable interface is the service provider interface (SPI) for PKCS #11 cryptographic services that are provided Oracle and third-party developers. Providers are user-level libraries that are implemented through encryption services available from hardware or software.
- **Scheduler and load balancer.** The operating system kernel includes software that is responsible for coordinating use, load balancing, and dispatching of cryptographic service requests.
- **Kernel programmer interface.** This interface provides kernel-level consumers with access to cryptographic services.
- **Service provider interface.** The interface to be used by providers of kernel-level cryptographic services that are implemented in hardware or software.
- **Hardware and software cryptographic providers.** Kernel-level cryptographic services that utilize software algorithms, hardware accelerator boards, or on-chip cryptographic capabilities (Oracle servers with SPARC T3 processors only).
- **Kernel cryptographic framework daemon.** The private daemon responsible for managing system resources for cryptographic operations. The daemon also verifies cryptographic providers.
- **Module verification library.** A private library used to verify the integrity and authenticity of all binaries that the cryptographic framework imports.
- **`elfsign`.** A utility offered to third-party providers of cryptographic services. The `elfsign` utility is used to request certificates from Sun. It also enables providers to actually sign the binaries, that is, `elf` objects that plug into the Oracle Solaris Cryptographic Framework.
- **`cryptoadm`.** A user-level command for managing cryptographic services, such as disabling and enabling cryptographic mechanisms according to security policy.

Keys to Working with the Oracle Solaris Cryptographic Framework

In general, four types of applications can plug into the Oracle Solaris Cryptographic Framework: user-level consumers, user-level providers, kernel-level consumers, and kernel-level providers. Developers creating applications should keep the following in mind.

- **User-level consumers.** When developing a user-level consumer, be sure to include the `security/cryptoki.h` file and link with the `libpkcs11.so` shared object. All calls should be made through PKCS#11 interfaces, and libraries should not call the `C_Finalize()` function.
- **User-level providers.** Design the provider to stand alone and create a PKCS#11 Cryptoki implementation in a shared object that includes all necessary symbols. Ideally, provide an `a_fini()` routine for data cleanup. Once complete, obtain a certificate and use it to `elfsign` the binary, and package the shared object for distribution.
- **Kernel-level consumers.** When developing a kernel-level consumer, include the `sys/crypto/common.h` and `sys.crypto/api.h` files. All calls should be made through the kernel programming interface.
- **Kernel-level providers.** When developing a kernel-level consumer, include the `sys/crypto/common.h` and `sys.crypto/api.h` files. Be sure to import required routines for registering, unregistering, and providing status, and export routines that provide entry points for the kernel cryptographic framework, as well as data structures with descriptions of supported algorithms. Once the loadable kernel module is created, obtain a certificate and use it to `elfsign` the binary, and package the kernel module for distribution.

Appendix E provides a list of PKCS#11 functions supported by the cryptographic framework.

Java APIs

Java security technology includes a large set of APIs, tools, and implementations of commonly used security algorithms, mechanisms, and protocols. Providing a comprehensive security framework for writing applications, Java APIs are available for cryptography, public key infrastructure, secure communication, authentication, and access control. Table 12-2 lists the key Java security APIs.

TABLE 12-2. KEY JAVA SECURITY APIs

NAME	DESCRIPTION
Java Cryptography Architecture (JCA)	A framework for accessing cryptographic functions in Java
Java Generic Security Services (Java GSS-API)	A Java version of the GSS-API interface
Java Secure Socket Extension (JSSE)	A Java version of the SSL and TSL protocols for secure Internet communication
Java Public Key Infrastructure (Java PKI)	A Java version of the PKI interface
Java Simple Authentication and Security Layer (Java SASL)	Classes and interfaces for applications that use SASL mechanisms
Java GSS-API	A Java implementation of the GSS-API interface

See http://www.oracle.com/technetwork/java/javase/tech/index-jsp-136007.html for more information on Java APIs for security.

Oracle Solaris Key Management Framework

Developers designing solutions that employ PKI technologies can utilize several keystore systems to store PKI objects, such as OpenSSL, PKCS#11, and NSS. Each keystore presents a different programming interface, and does not provide PKI policy enforcement. The Oracle Solaris Key Management Framework provides tools and programming interfaces for managing public key infrastructure (PKI) objects.

An API layer enables developers to specify the type of keystore to use, and provides plug-in modules. Applications written to the Oracle Solaris Key Management Framework are not bound to a keystore system. A system-wide policy database is available to applications, enabling applications to assert a policy and ensure subsequent operations behave according to policy limitations. Policy definitions include rules for performing validations, requirements for key and extended key usage, trust anchor definitions, Online Certificate Status Protocol (OCSP) parameters, and Certificate Revocation List (CRL) DB parameters, such as location.

More information on the Oracle Solaris Key Management Framework can be found at http://download.oracle.com/docs/cd/E19253-01/817-0547/getjm/index.html.

Chapter 13 Internationalization and Localization

*Internationalizatio*n makes software portable between languages or regions, so that it does not have to be rewritten to be used elsewhere. This is accomplished through the use of interfaces that modify program behavior at runtime. *Localization* adapts software for specific languages or regions by utilizing online information to support a language or region, known as a *locale*. Internationalized software can be ported from one locale to another without change. Like HP-UX 11i v3, Oracle Solaris contains extensive internationalization and localization support and provides the infrastructure and interfaces needed to create internationalized software.

Overview

The Oracle Solaris internationalization architecture provides a flexible, pluggable method of handling input methods, character set encodings, codeset conversion, and other basic aspects of language services. Applications can be deployed in multiple language environments without knowing how input methods work or which codeset converter needs to be enabled, simply by following standard APIs. The codeset independent approach to globalization enables operation in both native language and Unicode locales. Language attributes can be customized, and converter tables can be changed as needed. A rich set of data converters ensures interoperability between various encodings and different third-party platforms.

- **Localization functions and environment variables.** HP-UX and Oracle Solaris include the concept of *locales*, explicit models and definitions of native language environments. The notion of a locale is explicitly defined and included in the library definitions of the ANSI C language standard. Because the HP and Oracle compilers support this standard, and provide a consistent set of localization functions and environment variables, migrating applications from HP-UX to Oracle Solaris should be straightforward.
- **Full Unicode 4.0 support.** Oracle Solaris includes Unicode 4.0 support. Extensive support for languages and locales is provided, including complex text layout environments needed to support Thai and Hindi, and bidirectional layout environments for languages such as Arabic and Hebrew. HP-UX 11i v3 provides support for approximately 35 languages and 183 locales, while Oracle Solaris supports 55 languages and 345 locales.
- **Codeset independence.** Many areas around the globe, such as those utilizing PC-Kanji in Japan, Big5 in Taiwan, and GBK in the People's Republic of China, rely on non-Extended UNIX Code (non-EUC) codeset support. To address these concerns, yet maintain maximum application portability, Oracle Solaris OS provides a Codeset Independence (CSI) framework that enables both EUC and non-EUC codeset support. The CSI framework aims to remove dependencies on specific codesets or encoding methods from Oracle Solaris libraries and commands. Codeset independence enables application and platform software developers to keep source code independent of encoding such as UTF-8, and also provides the ability to adopt new encoding without having to modify the source code. While HP-UX and Oracle Solaris support EUC and non-EUC codesets, implementations may differ, and applications

may or may not need porting to the Oracle Solaris CSI framework to support internationalization efforts. Table 13-1 lists the CSI-enabled libraries in Oracle Solaris.

TABLE 13-1. CSI-ENABLED LIBRARIES IN ORACLE SOLARIS

`csetcol()`	`csetno()`	`euclen()`	`getwidth()`
`csetlen()`	`euccol()`	`eucscol()`	`wcsetno()`

- **Internationalization APIs.** Oracle Solaris provides APIs for multibyte (file code) and wide character (process code) that can help keep track of maintaining proper character boundaries when using multibyte characters. These APIs include messaging functions, code conversion, regular expressions, the wide character class, locale queries, character classification and transliteration, character collation, monetary and date and time formatting, and more.
- **Service based approach.** Unlike other UNIX platforms, Oracle Solaris 10 uses a service-based approach to administer language services remotely across a global network, regardless of the client system. This client-independent approach enables system upgrades without changing client applications.

Encoding Methods

HP-UX 11i v3 supports Unicode 5.0 for applications running on 32- and 64-bit Intel® Itanium® processor-based platforms, as well as applications compiled in shared mode on 32- and 64-bit PA-RISC platforms. All other applications compiled in archived mode on previous HP-UX releases use Unicode 3.0.

Oracle Solaris supports Unicode 4.0. Unicode is fully compatible with the international standards ISO/IEC 10646-1:2000 and ISO/IEC 10646–2:2001, and contains all the same characters and encoding points as ISO/IEC 10646. Any implementation that conforms to Unicode also conforms to ISO/IEC 10646. The Unicode Standard provides additional information about the characters and their use and can be referenced online at: http://www.unicode.org/standard/standard.html.

Unicode provides a consistent way of encoding multilingual plain text. Applications that support Unicode often are capable of displaying multiple languages and scripts within the same document. Any of the following character encoding schemes can be used for Unicode encoding:

- **UTF-8.** A variable-length encoding form of Unicode that preserves ASCII character code values transparently. This form is used for file codes in Oracle Solaris Unicode locales.
- **UTF-16.** A 16-bit encoding form of Unicode.
- **UTF-32.** A fixed-length, 21-bit encoding form of Unicode usually represented in a 32-bit container or data type. This form is used for process codes (wide-character codes) in Oracle Solaris Unicode locales.

These three encoding schemes are used for internal data processing in Oracle Solaris. When data is stored to disk, it is stored only in UTF-8 format. HP-UX also stores file data in UTF-8. As a result, data encoded in UTF-8 format can be exchanged between the two platforms without endian problems.

While there are no mandated codesets within the X/Open UNIX standard other than ASCII, the standard requires platforms to meet certain specifications for character manipulation. Adhering to this provision ensures that programs are capable of handling the characters encountered within non-ASCII codesets on a system.

Table 13-2 lists the supported codesets and encoding methods in HP-UX 11i v3 and their counterparts in Oracle Solaris. Codeset conversion utilities provide alternatives should a specific encoding be unavailable. Note that Oracle Solaris supports UTF-8 codesets for Asian languages with Unicode 3.2. Appendix B contains a list of supported locales and codesets for HP-UX and Oracle Solaris.

TABLE 13-2. ENCODING METHODS

HP-UX	ORACLE SOLARIS 10
ISO8859-1 (Latin-1)	ISO8859-1 (Latin-1)
ISO8859-2 (Latin-2)	ISO8859-2 (Latin-2)
ISO8859-5 (Latin-Cyrillic)	ISO8859-5 (Latin-Cyrillic)
ISO8859-7 (Latin-Greek)	ISO8859-7 (Latin-Greek)
ISO8859-8 (Latin-Hebrew)	Not used, Solaris uses a different approach for the Hebrew language that employs bi-directional text capabilities
ISO8859-9 (Latin-Turkish) (Latin-5)	ISO8859-9 (Latin-Turkish) (Latin-5)
ISO8859-13 (Latin-5) (Baltic Rim)	ISO8859-13 (Latin-5) (Baltic Rim)
ISO8859-15 (Latin-9)	ISO8859-15 (Latin-9)
Simplified Chinese Extended UNIX Code (EUC)	Simplified Chinese Extended UNIX Code (GB2312)
GBK	Simplified Chinese GBK
GB18030-2000	Simplified Chinese GB18030-2000
Not available	Simplified Chinese UTF-8 (Unicode 3.2)
Traditional Chinese EUC	Traditional Chinese EUC locale (CNS11643-1992)
BIG5	Traditional Chinese BIG5
Not available	Traditional Chinese UTF-8 (Unicode 3.2)
Not available	Traditional Chinese (Hong Kong) BIG5-HKSCS

TABLE 13-2. ENCODING METHODS

HP-UX	ORACLE SOLARIS 10
Not available	Traditional Chinese (Hong Kong) UTF-8 (Unicode 3.2)
Not available	Korean EUC (KS X)
Not available	Korean UTF-8 (Unicode 3.2)
Shift JIS	Shift JIS
Not available	PC Kanji
Thai API Consortium/Thai Industrial Standard (TIS620)	Thai API Consortium/Thai Industrial Standard (TIS620)
Not available	KOI8-R

Input Methods

Oracle Solaris utilizes the Internet Intranet Input Method Framework (IIIMF) to support multiple language inputs and scripts. The IIIM server serves both IIIM and XIM (X input method) clients and is started by each individual user in all UTF-8 and Asian locales. In addition, users of European UTF-8 locales can input data using the Compose or dead keys. IIIMF supports various EMEA keyboard layout emulations such as French, Russian or Arabic.

A variety of Input Method Engines (IMEs) such as Chinese, Japanese, Korean, Thai, Indic, and Unicode (HEX/OCTAL) are available when the corresponding locale support is installed. The Input Method Preference Editor (`iiim-properties`) can be used to find existing IMEs.

HP-UX does not support IIIM, but does support existing input methods for locales utilizing UTF-8 encoding. For Asian languages, HP-UX utilizes the xjim (Japanese), xsim (simplified Chinese), xtim (traditional Chinese), and xkim (Korean) input servers. Developers can also find third-party products that support other input methods for both HP-UX and Oracle Solaris. Oracle Solaris also supports optional code table input methods. Table 13-3 lists available input methods for Asian languages on HP-UX and Oracle Solaris.

TABLE 13-3. ASIAN LANGUAGE INPUT METHODS

HP-UX 11I v3	ORACLE SOLARIS 10
JAPANESE INPUT METHODS	
ATOK X	ATOK for Solaris (ATOK17)
Not Supported	Wnn6
SIMPLIFIED CHINESE INPUT METHODS (USED FOR ZH, ZH_CN, ZH_CN.EUC, ZH.GBK, ZH_CN.GBK, ZH_CN.GB818030, ZH.UTF-8, AND ZH_CN.UTF-8 LOCALES)	
ABC IM Supports	NewQuanPin
ABC IM Supports	NewShuangPin
ShuangPin	ShuangPin
QuanPin	QuanPin
Not Supported	English_Chinese
WangMa Wubi	WangMa Wubi
SIMPLIFIED CHINESE INPUT METHODS (USED FOR ZH, ZH_CN, ZH_CN.EUC LOCALES ONLY)	
GB2312 NeiMa	GB2312
SIMPLIFIED CHINESE INPUT METHODS (USED FOR ZH.GBK AND ZH_CN.GBK LOCALES ONLY)	
Not Supported	GBK NeiMa
SIMPLIFIED CHINESE INPUT METHODS (USED FOR ZH_CN.GB818030, ZH.UTF-8, AND ZH-CN.UTF-8 LOCALES ONLY)	
GB18030 NeiMa	GB18030 NeiMa
TRADITIONAL CHINESE INPUT METHODS	
Not Supported	New ChuYin
ChuYin	ChuYin
Rapid TsangChieh	
Not Supported	Array
Not Supported	Boshiamy
Not Supported	DaYi
Not Supported	JianYi

TABLE 13-3. ASIAN LANGUAGE INPUT METHODS

HP-UX 11I v3	ORACLE SOLARIS 10
Not Supported	Cantonese
NeiMa (EUC, BIG5, BIG5-HKSCS)	NeiMa (EUC, BIG5, BIG5-HKSCS)
Not Supported	English-Chinese
Not Supported	Optional codetable input methods, such as PinYin
KOREAN INPUT METHODS	
Hangul	Hangul
HanJa	HanJa
ASCII	ASCII
HEX Code	Special Symbols

Codeset Converters

Both the HP-UX and Oracle Solaris platforms support a broad range of codeset converters. The `iconv` and `sdtconvtool` utilities in Oracle Solaris can be used for code conversions among the major codesets of many countries. These utilities are included in Oracle Solaris as part of Unicode locale support. In addition, the `geniconvtbl` utility enables user-defined code conversions, but there is no similar utility on HP-UX.

The `geniconvtbl` utility enhances the ability of an application to deal with incompatible data types, particularly data generated from proprietary or legacy applications. Modifications to or customizations of existing Oracle Solaris codeset conversions are also supported. In Oracle Solaris, user-defined code conversions can be used with both `iconv(1)` and `iconv(3)`. Once the user-defined code conversions are prepared and placed properly, users can use the code conversions from the `iconv`(1) utility and the `iconv`(3C) functions of both 32-bit and 64-bit Oracle Solaris operating system. See the `iconv` man pages for more information about codeset converters on both HP-UX and Oracle Solaris.

Oracle Solaris also offers an Auto Encoding Finder, `auto_ef`, which is a useful utility for global character handling. This utility provides an easy way to detect the encoding of a particular file or string, and helps users to determine various language character encodings. The `auto_ef` utility is also useful for simplifying the display of web pages that do not specify encoding information. Other uses for the utility include detecting encoding in search engines, knowledge databases, and machine translation tools.

Locales

Locales contain explicit models and definitions of native-language environments and are included in the library definitions of the ANSI C language standard. HP and Sun compilers support this standard and provide a consistent set of localization functions and environment variables, so migrating applications form HP-UX to Oracle Solaris should be straightforward. The C, or POSIX, locale is the system default locale for all POSIX-compliant systems. Both HP-UX and Oracle Solaris utilize the C locale as the system default.

Appendix C lists supported locales for Oracle Solaris and their corresponding names on HP-UX 11i. Note that many languages have multiple locales, depending on language, region, and codeset. Users should select the locale that contains the date and time formats, currency, and more, that correspond to the correct locale parameters.

While non ISO compliant short form locales, such as *de* and *ja*, are supported in Oracle Solaris 10, their use should be avoided for compatibility with future Oracle Solaris releases.

Message Catalogs

A key step in internationalization is to divide software into executable code and all the messages seen by users. The message strings are kept in a separate *message catalog* and then translated for use in a language or region. Users specify the locale to use at login and the software then displays the translated messages. Locale-specific conventions are followed for formatting and displaying date, time, currency, and other information.

HP-UX and Oracle Solaris both support the creation and access of message catalogs through common, standard interfaces and commands based on specifications from The Open Group (XG4). Both maintain executable code and text messages separately, although Oracle Solaris makes more extensive use of a message catalog system known as `gettext`.

X and Motif Applications

Languages such as Arabic, Hebrew, and Thai require Complex Text Layout (CTL) capabilities in supporting platforms. Oracle Solaris contains CTL extensions that enable the Motif APIs to support languages that require complex transformations between logical and physical text representations. CTL Motif provides character shaping, such as ligatures, diacritics, and segment ordering. Support for the transformations of static and dynamic text widgets is also provided, along with bidirectional text capability and tabbing for dynamic text widgets. The CTL support in Oracle Solaris means that developing applications for languages such as Arabic, Hebrew, or Thai requires few changes.

Appendix A C Library Mapping

TABLE A-1. C LIBRARY EQUIVALENCE FOR HP-UX 11I AND ORACLE SOLARIS

HP-UX 11i v3	ORACLE SOLARIS 10
libc.so	libc.so libaio.so libbsdmalloc.so libcrypt.so libform.so libmalloc.so libmapmalloc.so libmenu.so libmtmalloc.so libnsl.so libproc.so libresolv.so libsec.so libsocket.so libthread.so libtnfprobe.so
libcurses.so	libplot.so libvt.so
libCsup.so	libc.so
libI077.a	libc.so
libm.so	libc.so
libdcekt.so	libgss.so
libgss.so	libgss.so
libgssapi_krb5.so	libgss.so
libipv6.so	libsocket.so
libnsl.so	libmd5.so librac.so libnsl.so
librpcsvc.so	libnsl.so
librt.so	librt.so
libxnet.so	libsocket.so
libxti.so	libnsl.so

Appendix B API Differences

TABLE B-1. KEY API DIFFERENCES

DESCRIPTION	HP-UX 11i v3	ORACLE SOLARIS 10
FUNCTIONS THAT ARE DIFFERENT IN NAME, OR ARE ONLY AVAILABLE ON ONE PLATFORM		
Obtain information about a mounted file system	`statfs()` `fstatfs()`	`statvfs()` `fstatvfs()`
Read directory entries	`getdirentries()`	`getdents()`
Set user ID	`setuid()`	`setuid()` `seteuid()`
Set group ID	`setgid()`	`setgid()` `setegid()`
FUNCTIONS THAT ARE DIFFERENT IN NAME AND HAVE DIFFERENT ARGUMENTS		
Get or set a file's Access Control List (ACL)	`getacl()` `fgetacl()`	`acl()` `facl()`
Get audit information for a process	`getaudid()`	`getauid()`
Set audit information for a process	`setaudid()`	`setauid()`
Manipulate auditing settings	`getaudproc()` `setaudproc()`	`auditon()`
FUNCTIONS WITH DIFFERENT ARGUMENTS		
Get and set process limits	`ulimit()`	`ulimit()`
Shared memory operations	`shmdt()`	`shmdt()`

Appendix C Summary of Supported Locales

The following table details the countries and locales supported by HP-UX 11i v3 and Oracle Solaris. Note that for the HP-UX versions, the locale binaries provided are version 3 and only applications running on Intel Itanium processor-based platforms or running in shared mode on PA-RISC are supported.

TABLE C-1. SUPPORTED LOCALES LISTED BY COUNTRY

COUNTRY	HP-UX 11i v3	ORACLE SOLARIS 10
Albania	–	sq_AL.ISO8859-2
Albania	–	sq_AL.UTF-8
Algeria	ar_DZ.arabic8	–
Algeria	ar_DZ.utf8	–
Argentina	es_AR.iso88591	es_AR.ISO8859-1
Argentina	es_AR.iso885915	–
Argentina	es_AR.utf8	es_AR.UTF-8
Australia	–	en_AU.ISO8859-1
Australia	–	en_AU.UTF-8
Austria	–	de_AT.ISO8859-1
Austria	–	de_AT.ISO8859-15
Austria	–	de_AT.UTF-8
Belgium-Flemish	–	nl_BE.ISO8859-1
Belgium-Flemish	–	nl_BE.ISO8859-15
Belgium-Flemish	–	nl_BE.UTF-8
Belgium-Walloon	–	fr_BE.ISO8859-1
Belgium-Walloon	–	fr_BE.ISO8859-15
Belgium-Walloon	–	fr_BE.UTF-8
Bolivia	es_BO.iso88591	es_AR.ISO8859-1
Bolivia	es_BO.iso885915	–
Bolivia	es_BO.utf8	es_AR.UTF-8
Bosnia	–	sh_BA.ISO8859-2@bosnia
Bosnia	–	sh_BA.UTF-8@bosnia
Brazil	pt_BR.iso88591	pt_BR.ISO8859-1
Brazil	pt_BR.iso885915	–
Brazil	pt_BR.utf8	pt_BR.UTF-8
Bulgaria	bg_BG.iso88595	bg_BG.ISO8859-5

TABLE C-1. SUPPORTED LOCALES LISTED BY COUNTRY

COUNTRY	HP-UX 11i v3	ORACLE SOLARIS 10
Bulgaria	bg_BG.utf8	bg_BG.UTF-8
C (Default for UNIX systems, same as POSIX)	C	C
Canada	—	en_CA.ISO8859-1 (English)
Canada	—	en_CA.UTF-8 (English)
Canada	—	fr_CA.ISO8859-1 (French)
Canada	—	fr_CA.UTF-8 (French)
Chile	—	es_CL.ISO8859-1
Chile	—	es_CL.UTF-8
China	zh_CN.gb18030	zh_CN.GB18030
China	zh_CN.hp15CN	—
China	zh_CN.utf8	zh_CN.UTF-8
China	zh_CN.EUC	—
China	zh_CN.GBK	—
Colombia	—	es_CO.ISO8859-1
Colombia	—	es_CO.UTF-8
Costa Rica	—	es_CR.ISO8859-1
Costa Rica	—	es_CR.UTF-8
Croatia	—	hr_HR.ISO8859-2
Croatia	—	hr_HR.UTF-8
Cypress	—	el_CY.UTF-8
Czech Republic	cs_CZ.iso88592	cs_CZ.ISO8859-2
Czech Republic	cs_CZ.utf8	cs_CZ.UTF-8@euro
Denmark	da_DK.iso88591	da_DK.ISO8859—1
Denmark	da_DK.iso885915@euro	da_DK.ISO8859—15
Denmark	da_DK.roman8	—
Denmark	da_DK.utf8	da_DK.UTF—8
Dominican Republic	es_DO.iso88591	—
Dominican Republic	es_DO.iso885915	—
Dominican Republic	es_DO.utf8	—
Ecuador	—	es_EC.ISO8859-1
Ecuador	—	es_EC.UTF-8
Egypt	—	ar_EG.UTF-8
Egypt	—	ar
El Salvador	—	es_SV.ISO8859-1
El Salvador	—	es_SV.UTF-8

TABLE C-1. SUPPORTED LOCALES LISTED BY COUNTRY

COUNTRY	HP-UX 11i v3	ORACLE SOLARIS 10
Estonia	et_EE.iso885915	et_EE.ISO8859-15
Estonia	et_EE.iso88594	—
Estonia	et_EE.utf8	et_EE.UTF-8
Finland	—	fi_FI.ISO8859—1
Finland	—	fi_FI.ISO8859—15
Finland	—	fi_FI.UTF-8
France	—	fr_FR.ISO8859-1
France	—	fr_FR.ISO8859-15
France	—	fr_FR.UTF-8
Germany	de_DE.iso88591	de_DE.ISO8859-1
Germany	de_DE.iso885915@euro	de_DE.ISO8859-15
Germany	de_DE.iso88594	—
Germany	de_DE.roman8	—
Germany	de_DE.utf8	de_DE.UTF-8 (Unicode 4)
Great Britain	—	en_GB.ISO8859-1
Great Britain	—	.ISO8859-15 (euro)
Great Britain	—	en_GB.UTF-8
Greece	el_GR.greek8	—
Greece	el_GR.iso88597	el_GR.ISO8859-7
Greece	el_GR.utf8	el_GR.UTF-8
Guatemala	—	es_GT.ISO8859-1
Guatemala	—	es_GT.UTF-8
Honduras	es_HN.iso88591	—
Honduras	es_HN.iso885915	—
Honduras	es_HN.utf8	—
Hong Kong	zh_HK.hkbig5	zh_HK.BIG5HK
Hong Kong	zh_HK.utf8	zh_HK.UTF-8
Hungary	—	hu_HU.ISO8859-2
Hungary	—	hu_HU.UTF-8
Iceland	—	is_IS.ISO8859—1
Iceland	—	is_IS.UTF-8
India	—	hi_IN.UTF-8
Ireland	—	en_IE.ISO8859-1
Ireland	—	en_IE.ISO8859-15
Ireland	—	en_IE.UTF-8

TABLE C-1. SUPPORTED LOCALES LISTED BY COUNTRY

COUNTRY	HP-UX 11i v3	ORACLE SOLARIS 10
Israel	—	he
Israel	—	he_IL.UTF-8
Italy	—	it_IT.ISO8859-1
Italy	—	it_IT.ISO8859-15
Italy	—	it_IT.UTF-8
Japan	—	ja
Japan	—	ja_JP.eucJP
Japan	—	ja_JP.PCK
Japan	—	ja_JP.UTF-8
Korea	—	ko_KR.EUC
Korea	—	ko_KR.UTF-8
Latvia	lv_LV.iso885913	lv_LV.ISO8859-13
Latvia	lv_LV.iso88594	—
Latvia	lv_LV.utf8	lv_LV.UTF-8
Lithuania	lt_LT.iso885913	lt_LT.ISO8859-13
Lithuania	lt_LT.iso88594	—
Lithuania	lt_LT.utf8	lt_LT.UTF-8
Luxembourg	—	de_LU.UTF-8
Macedonia	—	mk_MK.ISO8859-5
Macedonia	—	mk_MK.UTF-8
Malta	—	mt_MT.UTF-8 (Maltese)
Malta	—	en_MT.UTF-8 (English)
Mexico	—	es_MX.ISO8859—1
Mexico	—	es_MX.UTF-8
Netherlands	—	nl_NL.ISO8859-1
Netherlands	—	nl_NL.ISO8859-15
Netherlands	—	nl_NL.UTF-8
New Zealand	—	en_NZ.ISO8859-1
New Zealand	—	en_NZ.UTF-8
Nicaragua	—	es_NI.ISO8859-1
Nicaragua	—	es_NI.UTF-8
Norway	—	nn_NO.UTF-8
Norway	—	no_NO.ISO8859—1@bokmal
Norway	—	no_NO.ISO8859—1@nyorsk
Norway	no_NO.utf8	no_NO.UTF-8

TABLE C-1. SUPPORTED LOCALES LISTED BY COUNTRY

COUNTRY	HP-UX 11i v3	ORACLE SOLARIS 10
Norway	no_NO.iso88591	–
Norway	no_NO.iso885915@euro	–
Panama	–	es_PA.ISO8859-1
Panama	–	es_PA.UTF-8
Paraguay	–	es_PY.ISO8859-1
Paraguay	–	es_PY.UTF-8
Peru	–	es_PE.ISO8859-1
Peru	–	es_PE.UTF-8
Poland	pl_PL.iso88592	pl_PL.ISO8859-2
Poland	pl_PL.utf8	pl_PL.UTF-8
Portugal	pt_PT.iso88591	pt_PT.ISO8859-1
Portugal	pt_PT.iso885915@euro	pt_PT.ISO8859-15
Portugal	pt_PT.roman8	–
Portugal	pt_PT.utf8	pt_PT.UTF-8
POSIX	POSIX	POSIX
Romania	ro_RO.iso88592	ro_RO.ISO8859-2
Romania	ro_RO.utf8	ro_RO.UTF-8
Russian Federation	ru_RU.cp1251	–
Russian Federation	ru_RU.iso88595	ru_RU.ISO8859-5
Russian Federation	ru_RU.koi8r	ru_RU.KOI8-R
Russian Federation	ru_RU.utf8	ru_RU.UTF-8
Saudi Arabia	ar_SA.arabic8	–
Saudi Arabia	ar_SA.iso88596	–
Saudi Arabia	ar_SA.utf8	Ar_SA.UTF-8
Serbia	–	sr_YU.ISO8859-5
Serbia and Montenegro	–	sr_CS.UTF-8
Slovakia	sk_SK.iso88592	sk_SK.ISO8859-2
Slovakia	sk_SK.utf8	sk_SK.UTF-8
Slovenia	sl_SI.iso88592	sl_SI.ISO8859-2
Slovenia	sl_SI.utf8	sl_SI.UTF-8
Spain	–	ca_ES.ISO8859-1 (Catalan)
Spain	–	ca_ES.UTF-8 (Catalan)
Spain	–	ca_ES.ISO8859-15 (Catalan)
Spain	–	es_ES.ISO8859-1 (Spanish)
Spain	–	es_ES.ISO8859-15 (Spanish)

TABLE C-1. SUPPORTED LOCALES LISTED BY COUNTRY

COUNTRY	HP-UX 11i v3	ORACLE SOLARIS 10
Spain	—	es_ES.UTF-8 (Spanish)
Sweden	sv_SE.iso88591	sv_SE.ISO8859—1
Sweden	sv_SE.iso885915@euro	sv_SE.ISO8859—15
Sweden	sv_SE.roman8	—
Sweden	sv_SE.utf8	sv_SE.UTF-8
Switzerland	—	de_CH.ISO8859-1 (German)
Switzerland	—	de_CH.UTF-8 (German)
Switzerland	—	fr_CH.ISO8859-1 (French)
Switzerland	—	fr_CH.UTF-8 (French)
Taiwan	zh_TW.eucTW	zh_TW.EUC
Taiwan	zh_TW.big5	zh_TW.BIG5
Taiwan	zh_TW.utf8	zh_TW.UTF-8
Taiwan	zh_TW.ccdc	—
Thailand	th_TH.tis620	th_TH.TIS620
Thailand	—	th_TH.UTF-8
Turkey	tr_TR.iso88599	tr_TR.ISO8859-9
Turkey	tr_TR.turkish8	—
Turkey	tr_TR.utf8	tr_TR.UTF-8
Ukraine	uk_UA.cp1251	—
Ukraine	uk_UA.utf8	—
United Kingdom	en_GB.iso88591	—
United Kingdom	en_GB.iso885915@euro	—
United Kingdom	en_GB.roman8	—
United Kingdom	en_GB.utf8	—
United States	en_US.iso88591	en_US.ISO8859-1
United States	en_US.roman8	—
United States	en_US.utf8	en_US.UTF-8
United States	—	en_US.ISO8859-15
United States	es_US.iso88591 (Spanish)	—
United States	es_US.iso885915 (Spanish)	—
United States	es_US.utf8 (Spanish)	—
Uruguay	—	es_UY.ISO8859-1
Uruguay	—	es_UY.UTF-8
Venezuela	—	es_VE.ISO8859-1
Venezuela	—	es_VE.UTF-8

Appendix D Privileges Interfaces

TABLE D-1. PRIVILEGES INTERFACES

PURPOSE	FUNCTIONS	DESCRIPTION
Get and set privilege sets	`getppriv(2)`	Get a privilege set.
	`setppriv(2)`	Set a privilege set.
	`priv_set(3C)`	A wrapper for the `setppriv(2)` function.
	`priv_ineffect(3)`	A wrapper for the `getppriv(2)` function.
Identify and translate privileges	`priv_str_to_set(3C)`	Maps a privilege specification to a privilege set.
	`priv_set_to_str(3C)`	Converts the privilege set to a sequence of privileges.
	`priv_getbyname(3C)`	Map a privilege name to a number.
	`priv_getbynum(3C)`	Map privilege numbers to names.
	`priv_getsetbyname(3C)`	Map privilege set names to a number.
	`priv_getsetbynum(3C)`	Map privilege numbers to names.
Manipulate privilege sets	`priv_allocset(3C)`	Allocates memory for a privilege set.
	`priv_freeset(3C)`	Free the storage allocated by the `priv_allocset()` function.
	`priv_emptyset(3C)`	Clears all privileges.
	`priv_fillset(3C)`	Asserts all privileges.
	`priv_isemptyset(3C)`	Checks whether an argument is an empty set.
	`priv_isfullset(3C)`	Checks whether the argument is a full set (all bits set).
	`priv_isequalset(3C)`	Checks whether two privilege sets are equal.
	`priv_issubset(3C)`	Checks whether a privilege set is a subset of another set.
	`priv_intersect(3C)`	Intersects two sets and returns the result.
	`priv_union(3C)`	Takes the union of two sets and returns the results.
	`priv_inverse(3C)`	Inverts a privilege set.
	`priv_addset(3C)`	Adds the named privilege to a specified set.
	`priv_copyset(3C)`	Copies a privilege set.
	`priv_delset(3C)`	Removes the named privilege from a specified set.
	`priv_ismember(3C)`	Checks whether the names privilege is a member of a set.
Get and set process flags	`getpflags(2)`	Get process flags.
	`setpflags(2)`	Set process flags.

Appendix E Cryptographic Functions

TABLE E-1. ORACLE SOLARIS CRYPTOGRAPHIC FRAMEWORK FUNCTIONS

CATEGORY	FUNCTIONS

General Purpose	`C_Initialize()`
	`C_Finalize()`
	`C_GetInfo()`
	`C_GetFunctionList()`

Session Management	`C_GetSlotList()`
	`C_GetSlotInfo()`
	`C_GetMechanismList()`
	`C_GetMechanismInfo()`
	`C_SetPIN()`

Encryption and Decryption	`C_EncryptInit()`
	`C_Encrypt()`
	`C_EncryptUpdate()`
	`C_EncryptFinal()`
	`C_DecryptInit()`
	`C_Decrypt()`
	`C_DecryptUpdate()`
	`C_DecryptFinal()`

Message Digesting	`C_DigestInit()`
	`C_Digest()`
	`C_DigestKey()`
	`C_DigestUpdate()`
	`C_DigestFinal()`

Signing and Applying MAC	`C_Sign()`
	`C_SignInit()`
	`C_SignUpdate()`
	`C_SignFinal()`
	`C_SignRecoverInit()`
	`C_SignRecover()`

Signature Verification	`C_Verify()`
	`C_VerifyInit()`
	`C_VerifyUpdate()`
	`C_VerifyFinal()`
	`C_VerifyRecoverInit()`
	`C_VerifyRecover()`

Dual-Purpose Cryptographic Functions	`C_DigestEncryptUpdate()`
	`C_DecryptDigestUpdate()`
	`C_SignEncryptUpdate()`
	`C_DecryptVerifyUpdate()`

Random Number Generation	`C_SeedRandom()` `C_GenerateRandom()`

Object Management	`C_CreateOject()`
	`C_DestroyObject()`
	`C_CopyObject()`
	`C_FindObjects()`
	`C_FindObjectsInit()`
	`C_FindObjectsFinal()`
	`C_GetAttributeValue()`
	`C_SetAttributeValue()`

Key Management	`C_GenerateKey()`
	`C_GenerateKeyPair()`
	`C_DeriveKey()`

Appendix F Command Comparison Summary

Table F-1 provides a summary of key command differences between HP-UX 11i v3 and Oracle Solaris that are of concern to developers. Detailed information on these and other commands can be found in the Oracle Solaris man pages.

TABLE F-1. KEY COMMAND DIFFERENCES

COMMAND	PURPOSE	OPTIONS NOT SUPPORTED ON ORACLE SOLARIS 10	ORACLE SOLARIS 10 ADDITIONS	NOTES ON KEY DIFFERENCES
`asa`	Convert FORTRAN carriage-control output to printable form		`-f`	—`f` indicates each file should start on a new page.
`at`	Execute commands at a later time	`-d`	`-c, -k, -s`	Additional options specify the shell used to execute the job.
`awk`	Pattern scanning and processing	`-v`		Oracle Solaris supports one file with the —`f` option (HP-UX supports up to 100 files).
`batch`	Execute commands at a later time		`-p`	—`p` specifies the project for the run.
`bc`	Arbitrary precision arithmetic language			
`cat`	Concatenate and display files	`-r`		
`chgrp`	Change file group ownership		`-f`	—`f` suppresses error reporting.
`chown`	Change file ownership		`-f`	—`f` suppresses error reporting.
`cksum`	Write file checksums and sizes			
`compress`	Compress and uncompress files	`-d, -z`		On Oracle Solaris `uncompress` is equivalent to `compress` —`d`.
`csplit`	Split files based on context			
`delta`	Make a change to an SCCS file		`-d`	—`d` uses `diff(1)` rather than `bdiff(1)` for comparisons.
`env`	Set the environment for command execution			

`expr`	Evaluate arguments as an expression			Oracles Solaris supports the `length`, `match`, and `substr` operators only on x86 platforms, and does not support the `match` operator on SPARC or x86 platforms.
`false`	Provide truth values			
`gencat`	Generate a formatted message catalog	`-l`		
`get`	Retrieve a version of an SCCS file	`-w`	`-G`	–`G` specifies a new name for the retrieved file.
`getconf`	Get configuration values		`-a`	–`a` writes the names of the current system configuration variables to standard output.
`grep`	Search a file for a pattern			
`conv`	Code set conversion utility			
`ipcrm`	Remove a message queue, semaphore set, or shared memory ID		`-z`	–`z` specifies a zone.
`ipcs`	Report inter-process communication facilities status	`-C, -N`	`-A, -D, -i, -j, -z, -Z`	–`A` uses all print options (equivalent to -b, -c, -i, -J, -o, -p, and –t). `-D` displays contents of messages of a given type. `-i` prints the number of ISM attaches to shared memory segments. `-j` prints the creator's project. `-z` prints information about facilities associated with the specified zone. –`Z` prints information about all zones (when running in the global zone).
`join`	Form a join of the two relations			
`kill`	Terminate or signal a process			Oracle Solaris supports an optional exit status for the –`l` option.
`locale`	Get locale-specific information	`-A, -pa32, -pa64`		Oracle Solaris does not support the –`A`, `-pa32`, and –`pa64` options used on HP-UX for Itanium and PA-RISC systems.
`m4`	Macro processor			On Oracle Solaris, the –`B`, `-H`, `-S`, and `-T` options require a space between the option and its specified value.

`mkdir`	Make directories			
`mv`	Move files	`-e`		
`nice`	Invoke a command with an altered scheduling priority			
`nm`	Print the name list of an object file	`-d, -N, -q`	`-D, -R`	—`D` displays the symbol table used by the link-editor `ld.so.1` —`R` prints the archive name followed by the object file and symbol name.
`patch`	Apply changes to files		`-u`	—`u` interprets the patch file as a unified context difference.
`read`	Read a line from standard input			
`sed`	Stream editor			
`sleep`	Suspend execution for an interval			
`strip`	Strip symbol table, debugging, and line number information from an object file	`-r, -U`		
`test`	Evaluate conditions			Oracle Solaris supports the primary operators found in the HP-UX version of the command, adds a few options, and supports a richer set of conditions.
`true`	Provide truth values			
`ulimit`	Set or get limitations on the system resources available to the current shell and its descendents			HP-UX only provides a `ulimit(2)` C interface. Oracle Solaris also provides a user level command `ulimit(1)`.
`wait`	Await process completion		`jobid`	HP-UX and Oracle Solaris support the specification of a process ID for which the utility is to wait for termination. Oracle Solaris also supports the specification of a job control ID that identifies a background process group for which to wait.
`xargs`	Construct and argument list and invoke a command			
`yacc`	Yet another compiler-compiler			`yacc` is not supported on HP-UX.

Appendix G Resources

Additional information and developer resources can be found in the references listed in Table G-1.

TABLE G-1. ADDITIONAL READING

ORACLE SOLARIS	
Oracle Solaris	http://www.oracle.com/solaris
Oracle Solaris 10 Documentation	http://download.oracle.com/docs/cd/E19253-01/index.html
Oracle Solaris Tunable Parameters Reference Manual	http://download.oracle.com/docs/cd/E19253-01/817-0404/817-0404.pdf
Oracle Solaris 10 What's New	http://download.oracle.com/docs/cd/E19253-01/817-0547/index.html
Oracle Solaris Cluster	http://www.oracle.com/technetwork/server-storage/solaris-cluster/index.html
PROGRAMMING REFERENCES	
Oracle Solaris 10 Documentation Set	http://www.oracle.com/technetwork/documentation/solaris-10-192992.html
Oracle Solaris Studio Portal	http://www.oracle.com/technetwork/server-storage/solarisstudio/overview/index.html
Oracle Solaris Studio Documentation	http://www.oracle.com/technetwork/server-storage/solarisstudio/documentation/oss-122-docs-169726.html
Linker and Libraries Guide	http://download.oracle.com/docs/cd/E19082-01/819-0690/book-info/index.html
Oracle Solaris Security for Developers Guide	http://download.oracle.com/docs/cd/E19253-01/816-4863/816-4863.pdf
Programming Interfaces Guide	http://download.oracle.com/docs/cd/E19253-01/817-4415/817-4415.pdf
Writing Device Drivers	http://download.oracle.com/docs/cd/E19963-01/html/819-3196/ddidkisvc-29227.html
"Developing Enterprise Applications with Oracle Solaris Studio" white paper	http://www.oracle.com/technetwork/server-storage/solarisstudio/documentation/oss-enterprise-apps-170707.pdf
"Examine MPI Applications with the Oracle Solaris Studio Performance Analyzer", white paper	http://www.oracle.com/technetwork/server-storage/solarisstudio/documentation/oss-performance-analyzer-177582.pdf
"Parallel Programming with Oracle Developer Tools" white paper	http://www.oracle.com/technetwork/systems/parallel-programming-oracle-develop-149971.pdf
"Taking Advantage of OpenMP 3.0 Tasking with Oracle Solaris Studio" white paper	http://www.oracle.com/technetwork/server-storage/solarisstudio/documentation/oss-openmp-30-wp-183987.pdf

JAVA	
Java Technology	http://www.oracle.com/java
JavaFX Platform	http://download.oracle.com/javafx/2.0/overview/jfxpub-overview.htm
Java Security	http://www.oracle.com/technetwork/java/javase/tech/index-jsp-136007.html
Java SE Security Documentation	http://download.oracle.com/javase/6/docs/technotes/guides/security/index.html
ORACLE SOLARIS SERVICE MANAGEMENT FACILITY	
“How to Create an Oracle Solaris Service Management Facility Manifest” white paper	http://www.oracle.com/technetwork/server-storage/solaris/solaris-smf-manifest-wp-167902.pdf
“Management of Systems and Services Made Simple with the Oracle Solaris Service Management Facility” white paper	http://www.oracle.com/technetwork/server-storage/solaris/solaris-smf-wp-167901.pdf
INTERNATIONALIZATION	
Oracle Solaris 10 Supported Locales	http://www.oracle.com/technetwork/systems/articles/solaris-10-ur2-locales-142856.html - north-americas
International Language Environments Guide	http://download.oracle.com/docs/cd/E19253-01/817-2521/index.html
Korean Solaris User's Guide	http://download.oracle.com/docs/cd/E19253-01/817-2522/index.html
Simplified Chinese Solaris User's Guide	http://download.oracle.com/docs/cd/E19253-01/817-2523/index.html
Traditional Chinese Solaris User's Guide	http://download.oracle.com/docs/cd/E19253-01/817-2524/index.html

Appendix H Glossary

ACAP
Application Configuration Access Protocol.

Access Control List (ACL)
A file containing a list of principals with certain access permissions. Typically, a server consults an access control list to verify that a client has permission to use its services.

Agent Builder
A component of Oracle Solaris Cluster that automates the creation of a data service.

API
Application programming interface.

appcert
A utility that examines an application's conformance to the Oracle Solaris Application Binary Interface. Use of the appcert utility can help identify potential binary compatibility issues when porting applications to Oracle Solaris.

Authentication
A security service that verifies a claimed identity.

Authorization
The process of determining whether a user can use service, which objects the user can access, and the type of access allowed.

Big Endian
An architecture that stores the most-significant byte of data first. Oracle Solaris uses a Big Endian architecture on SPARC processor-based systems and a Little Endian architecture on x86 platforms.

Bourne shell
The default shell in Oracle Solaris 10. The shell is found in `/usr/bin/sh`.

Chip-Multithreading Technology
Multithreaded processor technology that enables each processor core to switch between multiple threads on each clock cycle.

CMT
Chip-Multithreading Technology.

Consumer
An application, library, or kernel module that uses system resources.

Context
A state of trust between two applications.

Data transformation
The process of converting data from one format to another.

dbx, dbxtool
A scriptable, multithread-aware debugger for applications using Oracle Solaris or POSIX threads. The graphical version is `dbxtool`.

DDI/DKI
Device Driver Interface/Driver-Kernel Interface. Interfaces that standardize interactions between device drivers and the operating system kernel, device hardware, and boot and configuration software.

Discover
A tool that detects and reports memory access errors in a running application.

DLight
A tool that unifies application and system profiling using Oracle Solaris DTrace technology.

dmake
Distributed Make utility that parses makefiles and determines which targets can be built concurrently. The build of those targets is distributed over a number of hosts.

DTrace
See Oracle Solaris DTrace.

ETL utilities
Extract, Transform, and Load utilities, tools that take a wide array of formats and convert them into Structured Query Language (SQL) for relational database management systems.

GSS-API
The Generic Security Service Application Programming Interface. A network layer providing support for various modular security services. GSS-API provides security authentication, integrity, and confidentiality services, and allows maximum application portability with regard to security.

Hard limit
A resource consumption limit set by the operating system or processes with special privileges.

Hybrid Storage Pool
A combination of disk drives and Flash devices that work together to minimize the impact of disk latencies and improve application performance. Flash devices handle certain types of I/O while hard disk drives store massive data sets. Hybrid Storage Pools are enabled by Oracle Solaris ZFS.

IIIMF
Internet Intranet Input Method Framework.

IMAP
Internet Access Message Protocol.

Internationalization
Technology that makes software portable across languages and regions.

Java Platform, Enterprise Edition (Java EE)
A toolkit that builds on Java SE and adds an application server, Web server, Java 2 Platform, Enterprise Edition API, support for JavaBeans, Java servlets API, and JavaServer Pages (JSP) technology.

Java Platform, Standard Edition (Java SE)
A toolkit for developing Java applications. The toolkit includes a compiler, runtime environment, and core API.

JavaFX
A platform that provides a rich set of graphics and media APIs with high-performance hardware-accelerated graphics and media engines that simplify the development of data-driven enterprise client applications.

Korn shell
Oracle Solaris supports `ksh88`. The shell is located in `/usr/bin/ksh`.

Lazy loading
Enables the loading of a dependency to be delayed until the function is first referenced.

LDAP
Lightweight Directory Access Protocol.

LDI
Layered Driver Interface. An extension of the DDI/DKI that enables a kernel module to access other devices in the system.

Little Endian
An architecture that stores the least-significant byte of data first. Oracle Solaris uses a Little Endian architecture on x86 systems and a Big Endian architecture on SPARC processor-based platforms.

Locale
A language or region.

Localization
Technology that adapts software for specific languages or regions by utilizing online information to support a language or region, know as a locale.

maxfiles
A soft limit that specifies the file limit per process.

maxfiles_lim
A hard limit that specifies the file limit per process.

maxuprc
The maximum number of user processes allowed.

MPO
Memory Placement Optimization.

NetBeans IDE
A free and open source software development tool for creating Web, desktop, and mobile applications.

Oracle JDeveloper
A free, integrated development environment that simplifies the creation of Java-based SOA applications and user interfaces.

Oracle Solaris Cluster
A high availability solution for Oracle Solaris that is integrated at the kernel level. It monitors servers, storage, network components, operating system, virtual machines, and applications. Recovery actions are based on policies and application specifications.

Oracle Solaris Cryptographic Framework
A framework built into Oracle Solaris that provides kernel-level and user-level consumers with access to software-based or hardware-based cryptographic capabilities.

Oracle Solaris DTrace
A dynamic tracing facility built into Oracle Solaris that lets developers observe operating system and application behavior in real time.

Oracle Solaris Key Management Framework
A framework that provides tools and programming interfaces for managing PKI objects.

Oracle Solaris Service Management Facility
A facility introduced in Oracle Solaris 10 that simplifies service management and control.

Oracle Solaris Studio
A free, comprehensive C, C++, and Fortran tool suite for Oracle Solaris and Linux operating systems that accelerates the development of scalable, secure, and reliable enterprise applications.

Oracle Solaris ZFS
A 128-bit file system that integrates volume management and provides virtually unlimited file system scalability.

Oracle VM
Scalable server virtualization software that supports Oracle and non-Oracle applications.

Package
A collection of files and directories that are required for a software product. On Oracle Solaris, applications are distributed for deployment in packages.

PAM.
See Pluggable authentication modules.

PKI
Public Key Infrastructure.

Pluggable authentication modules
A framework that provides authentication and related security services for managing accounts, sessions, and passwords.

POSIX
Portable Operating System Interface for UNIX. A set of standards that provide a well-defined system call interface for kernel facilities, as well as shell and utilities interfaces.

POSIX Threads
A standard API that defines a set of C programming language types, functions, and constants related to multithreaded programming.

Privilege
A discrete right that can be granted to an application.

pthreads
See POSIX Threads.

Runtime checking
A debugging feature integrated into Oracle Solaris Studio that automatically detects runtime errors, such as memory access and memory leaks. The debugger interrupts program execution and displays the relevant source code upon error detection.

Runtime linker
Software that provides library calls that can be used to locate and bind applications to shared libraries during execution.

SASL
The Simple Authentication and Security Layer, a framework that provides authentication services and optional integrity and confidentiality services to connection-based protocols. SALS is appropriate for applications that use IMAP, SMTP, ACAP, and LDAP protocols, as these all support SASL.

SMF
See Oracle Solaris Service Management Facility.

SMTP
Simple Mail Transport Protocol.

SOA
Service-oriented architecture.

Soft limit
A resource consumption limit that can be changed by a process. A soft limit must remain less than or equal to the hard limit in Oracle Solaris.

Sun Performance Library
A set of optimized, high-speed mathematical subroutines for solving linear algebra and other numerically-intensive problems.

sysdef
A command that outputs the current system definition in tabular form.

Thread Analyzer
A tool that helps identify common issues in multithreaded code by analyzing program execution across multiple threads. Thread Analyzer is particularly helpful in detecting data race and deadlock conditions.

Trusted Extensions
An optional layer of secure label technology in Oracle Solaris that allows data security policies to be separated from data ownership. Multilevel data access policies support compliance goals.

UFS
UNIX File System, the default file system in Oracle Solaris 10.

ulimit
A command to set or get limitations on the system resources available to the current shell and its descendents.

umask
The file mode creation mask. Oracle Solaris sets a default umask of 022 in the `/etc/profile` file.

Uncover
A code coverage tool that facilitates the identification of major functional areas within binaries that are not being tested.

Unicode
An industry standard that specifies a consistent way of encoding multilingual plain text.

UTF-8
A variable-length encoding form of Unicode that preserves ASCII character code values transparently. It is used for file codes in Oracle Solaris Unicode locales.

UTF-16
A 16-bit encoding form of Unicode.

UTF-32
A fixed-length, 21-bit encoding form of Unicode usually represented in a 32-bit container or data type. It is used for process codes (wide-character code) in Oracle Solaris Unicode locales.

ZFS
See Oracle Solaris ZFS.

HP-UX to Oracle Solaris Porting Guide
August 2011, Version 1.0

Oracle Corporation
World Headquarters
500 Oracle Parkway
Redwood Shores, CA 94065
U.S.A.

Worldwide Inquiries:
Phone: +1.650.506.7000
Fax: +1.650.506.7200

oracle.com

Oracle is committed to developing practices and products that help protect the environment

Hardware and Software, Engineered to Work Together