ORACLE
SOLARIS

An Oracle Technical White Paper
July 2012

Red Hat Enterprise Linux to Oracle Solaris Porting Guide

ORACLE

What's New in Oracle Solaris 11? 4
Oracle Portability and Compatibility Guarantee 5
Similarities and Differences Between Oracle Solaris and Linux 5
Tools and Development Environment 6
GNU Utilities and Popular Developer Tools 6
Availability of Open Source Software on the Oracle Solaris Platform7
Advantages of Porting to Oracle Solaris 11 7
Recommended Strategy 9
Assessment of the Application Porting Effort 10
Limiting the Scope 10
Classification of Code 11
Scripts and Other Portable Components 11
Build Environment Dependencies 11
Assess Food Chain Dependencies 12
Data Portability 13
Data Migration Considerations 13
Data Portability, Well-Known Issues, and Solutions 13
Application Verification 15
Code Coverage 15
Memory Leak Detection 16
Supported Processor Architectures (Linux Versus Oracle Solaris) .. 17
Storage Order and Alignment 17
Default Alignment and Recommendations 17
Data Structures and Sizes 18
Compiler Options and Portability of Code 19
Byte Ordering 20
Data Conversion for Interoperability 20
Low-Level Code, Bit-Level Operations 21
System APIs 21
System Call Mapping 21
Reducing Migration Costs 29
Oracle Solaris Studio 29
Oracle Solaris Studio Components 29
Standards Adherence 30
Useful Tools 30

GNU Compiler Collection (GCC) Versus Oracle Solaris Studio Compiler Option Mapping 31
Dynamic Linking Using Hardware-Specific Performance Libraries 35
Open Source Software Libraries 36
Debugging Applications 47
Kernel Debuggers 47
Source-Level Debugging with dbx 47
Tools for Addressing Specific Issues 48
Identifying Issues Using DTrace 48
POSIX Compliance 50
Linux Threading Model 50
Selecting the Thread Implementation 50
Oracle Solaris Threading Model 51
Differences Between the Oracle Solaris and Linux Threading Models 51
Signals in Threaded Applications 54
Getting the Most out of Oracle Solaris 11 54
Support for Latest Hardware Technologies 54
OpenMP Support 55
Auto Parallelization and Compile-Time Optimizations 55
Addressing Multithreaded Application Issues 56
Using Thread Analyzer 56
Detecting Race Conditions and Deadlocks 56
Migrating Kernel Modules and Device Drivers 58
Device Driver Interface and Driver-Kernel Interface 60
Necessary Compiler Options, Linker Options, and Linked Libraries 61
Best Practices for Porting Device Drivers and Kernel Modules 62
Security Interfaces for Developers 63
Physical Security 63
Delegate Minimal Privileges—Only as Appropriate 63
Oracle Solaris Trusted Extensions 64
Ensure Strong Defenses 64
Encryption Algorithms, Mechanisms, and Their Mapping 65
Using Hardware Accelerators and System-Provided Interfaces 67
User and Process Privileges and Permissions 69
Resource Controls and Runtime Limits 71
Resource Limits on RHEL 71

Resource Limits on Oracle Solaris 11 72
Migrating Scripts 72
Equivalent Commands, Options, and Alternatives on Oracle Solaris 11 72
Managing Services and Daemons on RHEL 78
RHEL Service Management 79
Managing and Controlling Service Dependencies 79
Oracle Solaris Service Management Facility (SMF) 80
Managing Services Through SMF 81
Managing Service Dependencies in Oracle Solaris 82
Migrating to SMF 82
Service Manifests 82
Leveraging SMF Facilities in Oracle Solaris 11 83
Pluggable Authentication Module (PAM) 84
PAM Configuration and Differences 85
Migrating Custom PAM Modules—Developer's Perspective 85
Differences in PAM Data Structures and Function Calls 88
Necessary Compiler Option, Linker Options, and Linked Libraries 89
Packaging on RHEL 90
Categories of RPM Packages 91
Packaging on Oracle Solaris 11 92
Preparing an Application for Packaging 92
Implementation Differences for Packaging 94
Package Administration and Managing Dependencies and Upgrades 94
Building a Package in Oracle Solaris 95
Appendix A Security and Privileges 99
Appendix B GCC to Oracle Solaris Studio Compiler Flag Mapping 107

Chapter 1 Introduction

The purpose of this document is to enable developers to resolve issues faced during the migration of Red Hat Enterprise Linux (RHEL) applications to Oracle Solaris 11. The document describes similarities and differences between the two environments in terms of architecture, system calls, tools, utilities, development environments, and operating system features. Wherever possible, it also provides pointed solutions and workarounds to address the specific porting issues that commonly arise due to implementation differences on these two platforms.

Using the information presented in this guide, developers should be able to tackle projects ranging from the smallest data conversion to the largest legacy native-code migration projects.

This document also includes best practices to help developers get the most out of their applications when running them on Oracle Solaris 11. Specific guidance is offered to help avoid some of the pitfalls that are common to migration projects.

In the interest of larger developer groups with varied development and functional requirements, as well as to maintain the general usefulness of this guide, the guide avoids going too deep into the specifics of a given problem. Instead, pointers are provided to direct readers to additional relevant information for further reading. Oracle strongly advises both novice users and those familiar with the Oracle Solaris operating system to use man pages to obtain accurate and detailed information about Oracle Solaris 11 and its features.

What's New in Oracle Solaris 11?

Oracle Solaris 11 delivers the industry's best cloud operating system. It is designed to meet the security, performance, and scalability requirements of cloud-based deployments. As the first fully virtualized operating system, Oracle Solaris 11 provides comprehensive, built-in virtualization capabilities for OS, network, and storage resources. It offers comprehensive management across the entire infrastructure—operating system, physical hardware, networking, and storage, as well as the virtualization layer. Oracle has made a strong commitment to Oracle Solaris on both SPARC and x86 systems. Oracle is working to help its customers understand its strong commitment and the importance of offering a single operating system that runs on both SPARC (RISC) and x86 (CISC) processors.

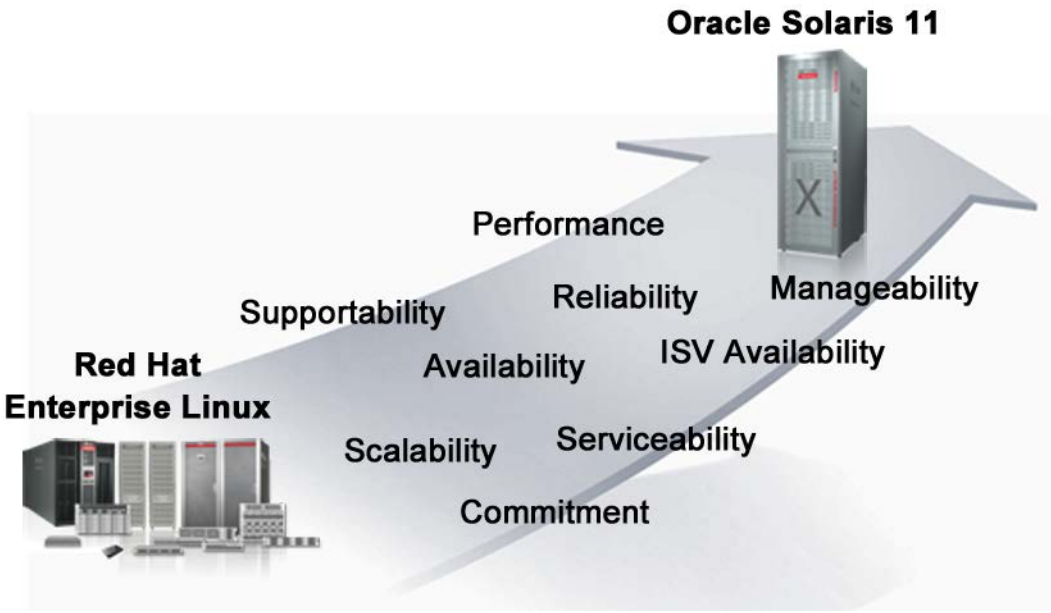

Figure 1-1. Oracle Solaris is designed to meet the security, performance, and scalability requirements of cloud-based deployments.

Oracle Portability and Compatibility Guarantee

Oracle Solaris has maintained binary compatibility between operating system releases for nearly a decade, enabling existing Oracle Solaris applications to run unmodified on Oracle Solaris 11. This means that Oracle Solaris applications developed over many years will run on Oracle Solaris 11 unchanged, taking full advantage of new and advanced Oracle Solaris features.

The Oracle Solaris Source Code Guarantee provides assurance that C and C++ applications developed and successfully compiled to run on either a SPARC or x86 platform will successfully compile and run on both platforms. Through Oracle's commitment to choice, organizations can confidently develop and deploy applications on whatever platform best suits their needs now and into the future. This guarantee helps protect long-term investments in software development, training, and IT staff skills.

To learn more about the Oracle Solaris Binary Compatibility Guarantee and the Oracle Solaris Source Code Guarantee, visit http://www.oracle.com/us/products/servers-storage/solaris/binary-app-guarantee-080255.pdf.

Similarities and Differences Between Oracle Solaris and Linux

The Single UNIX Specification (SUS) is an industry-standard description of C-language program and user command interfaces for a standard UNIX operating system. The SUS was developed to ensure that a program developed in one UNIX operating system would run in a somewhat different UNIX operating system. The specification is owned by The Open Group, an industry group that oversees UNIX certification and branding.

RHEL conforms to the Linux Standard Base 4.0 (LSB 4.0) specification. The LSB is based on the POSIX specification, the SUS, and several other open standards, but it extends them in certain areas.

Oracle Solaris supports IEEE Std 1003.1 and IEEE Std 1003.2, commonly known as POSIX.1 and POSIX.2, respectively. Specifically, Oracle Solaris 11 conforms to POSIX.1-2004 (POSIX.1-2001 update) and POSIX.1-2008 (POSIX.1-2004 update).

Since both the operating systems are POSIX-compliant (variants), there should be little difficulty porting code from RHEL to Oracle Solaris, as long as specific operating extensions are not used. If Linux-specific extensions are used, an equivalent Oracle Solaris extension will need to be used or the extensions will need to be manually ported in order to achieve the desired result.

Tools and Development Environment

RHEL is a standards-based UNIX operating system that provides a development environment very similar to Oracle Solaris 11. Most of the popular development and scripting tools used by Linux developers are also available on Oracle Solaris 11. For example, the default shell for `root` on both the operating systems is `bash` (version 4.1.x).

The k-shell (`ksh`) available on RHEL is `ksh93`, while the default shell for a *new user* on Oracle Solaris 11 is also `ksh93`. The availability of the same shell on both platforms makes it easier to migrate scripts from RHEL to Oracle Solaris 11.

Oracle Solaris 11 ships with hundreds of standard commands, tools, utilities, and services. These packages are built from the same open source code base that feeds Linux. The availability of these packages along with the normal Oracle Solaris functionalities is a benefit to Oracle Solaris users. Users can choose to use these Linux-like packages and, thereby, increase affinity with Linux. (Please note that environment variables, such as `PATH`, might need to be altered in order to use similar tools on both the platforms.)

Oracle Solaris 11 and RHEL are very similar in many ways; for example, they both have support for Oracle Solaris Studio software and GNU tools. Both use X Windows as the GUI for their desktop interface. It is worth noting that if the GNU family of compilers is used on RHEL and you would also like to continue to use the GNU compiler for the SPARC platform, GNU compilers and tools are also available on Oracle Solaris 11.

In short, since the shell and development tools are very similar on the two platforms, it is very easy to move from RHEL to Oracle Solaris as an end user or a developer.

GNU Utilities and Popular Developer Tools

Given that Oracle Solaris 11 already provides a very significant RHEL baseline in terms of: commands, tools, libraries, platform services, and software development environment, the task of porting an application to Oracle Solaris 11 boils down to migrating the actual native code that needs to be modified. If the same development tools and compilers are used on both sides, the tool-related complexities during porting become minimal. Oracle has made Oracle Solaris Studio 12.x compilers and tools available on both Oracle Solaris and RHEL platforms. It is also important to note that the

GNU Compiler Collection (GCC) and other GNU tools are also available on both RHEL and Oracle Solaris 11.

Availability of Open Source Software on the Oracle Solaris Platform

Most of the popular open source software is either already available on Oracle Solaris 11, or the ported binaries as well as source code are made available through various popular open source repositories, such as http://www.sunfreeware.com/, http://www.blastwave.org/, http://www.sourceforge.net, http://www.solaris4you.dk/, http://www.unixpackages.com/, and many more.

Advantages of Porting to Oracle Solaris 11

There are many advantages moving to Oracle Solaris 11.

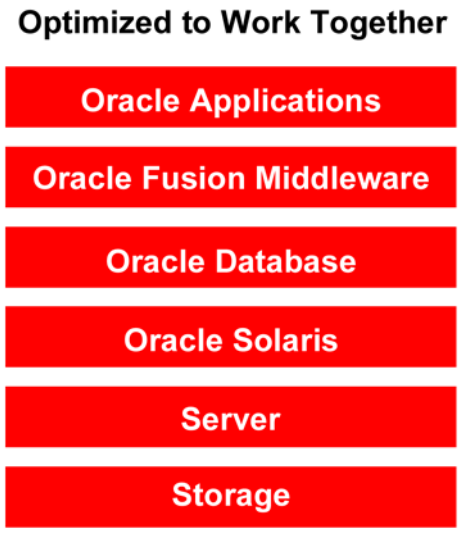

Figure 1-2. Oracle Solaris is part of Oracle's integrated hardware and software stack in which the components are optimized to work together to enhance performance, availability, and security.

The following list highlights some of the key capabilities of Oracle Solaris 11 that can make a big difference, particularly in demanding environment for large enterprise-wide deployments.

- Complete integrated stack (hardware to applications) optimized to work together
- Portability across architectures (SPARC, X64)
- Standards compliance
- Tools optimized to work best on the integrated stack
- Support for a wide range of hardware
- Infrastructure designed to scale on large systems *without touching code*
- Investments in new-feature development and technology advancements (R&D spends)

- Oracle's commitment for long-term investments
- End-to-end 24 x 7 support from Oracle (a single vender) for hardware to applications

Chapter 2 The Porting Process

Recommended Strategy

Due to both operating systems' conformance to standards and the availability of tools to aid migration, many times the migration process can be straightforward and hassle-free. However, it is important to plan the migration systematically, so as to isolate the areas where more effort might be needed.

Here are the steps that are followed in typical migration projects:

- Planning phase (making the right choices to minimize porting efforts):
  - Assessing application porting
  - Assessing the current environment
  - Estimating the migration effort
  - Choosing the right tools and build infrastructure
  - Incorporating Oracle-recommended tools and following best practices
- Execution phase (porting):
  - Module-wise porting
  - Unit testing
  - Integration testing
- Validation, testing, and certification phase:
  - Functional testing
  - System testing
  - Stress tests, soak tests, and long-haul tests
- Deployment phase (getting the most from the new platform):
  - Deployment of applications on the new architecture
  - Performance tuning

Oracle Solaris 11 has many built-in features that make it compatible with Linux, including commands, tools, utilities, and application programming interfaces (APIs). Oracle Solaris 11 ships with hundreds of popular Linux utilities and tools. Many popular GNU utilities, libraries, and applications are available on Oracle Solaris 11 as optional installable packages and can be installed based on specific application requirements. For example `gnu-coreutils`, `gnu-findutils`, `binutils`, `glib2`, `gtk2`, Perl, Bison, Ruby, Python, and other similar scripting languages as well as Tcl/Tk libraries, GNU Emacs, Apache HTTP server, the GCC compiler, and many other development tools are available on Oracle Solaris 11. It should be noted that for many of these utilities, the legacy Oracle Solaris versions as well as the GNU versions can be made available on the system, if you install some of these optional

packages on top of the default installation. Hence, you will have to carefully set the PATH environment in your development and deployment environments in order to invoke the correct utilities from the system.

From the previously mentioned migration phases, you estimaet the time required for completing most of the tasks because the amount of effort required does not very much with the target platform. For example, functional testing, soak testing, and stress testing, are usually fairly consistent across platforms. The area that can potentially pose the maximum risk is the complexities involved in porting the legacy native code. This risk can be minimized by spending sufficient time and effort during the planning phase.

Though both the operating systems are similar in many aspects, you might observe subtle differences when it comes to porting legacy native code. Hence, the effort required for the transition to Oracle Solaris 11 will greatly vary due to the composition of application components as well as the programming language and tools used by various application subcomponents. To arrive at an estimate for the porting effort, it is important to classify the various application subcomponents based on their implementation complexities.

Assessment of the Application Porting Effort

The most important part of migration process is the assessment of existing applications and the associated environment. This will allow you to create a risk list that can be used to identify any areas of the project that might require proof of concept to ensure that the project can be completed.

The outcome of the assessment will be a *risk list* and a *work breakdown* structure that details the amount of effort required to migrate the application modules and the associated environment. The work breakdown structure can then used to create a plan and to schedule various activities. During project execution, remember to allow sufficient time to follow Oracle recommended best practices as well as to time to re-architect some of the modules or to change the deployment strategy to get most from Oracle Solaris 11.

For custom, quickly evolving applications, it is very important to freeze a snapshot of the application source code and associated infrastructure to serve as a baseline for the migration activity. The following sections discuss best practices that can greatly help in identifying the overall migration effort and potential areas of risk.

Limiting the Scope

Understanding the composition of the code used by an application is a critical part of the planning process. Since many legacy applications are large (millions of lines of code), simply trying to understand the layout of the source tree and the types of files can become a complex task. Developers seldom remove old, unused code from the source code directory, and seldom are there different build instructions for each targeted deployment scenario.

As an application evolves, new functionality gets added, some business functionality becomes redundant or irrelevant, and some modules are no longer required for a given deployment scenario.

Hence, for a large application, understanding which files within the source distribution are actually getting used to build the application for the given deployment scenario can help limit the scope of the porting activity.

Classification of Code

Before starting the actual porting process, it is good to segregate the code based on the amount of migration effort required for each unit. This will allow you to estimate the overall effort required for the migration. For example, if 80 percent of the code is portable (for example, Java) and if 10 percent is scripts, only the remaining 10 percent of the code might have bigger porting challenges and need more attention.

The easiest way to arrive at this estimate is to segregate code based on the programming languages used for coding, and then evaluate each one of them separately for porting complexities. For example, as a thumb rule, code written in Java, Perl scripts, and shell scripts might present fewer challenges compared to native modules written in the C, C++, or Visual C programming languages. Needless to say, you might come across projects with exceptions. The porting of scripts is one such area which needs careful planning and assessment. The following section discusses in more detail the potential issues during script migration.

Scripts and Other Portable Components

The Perl utility is popular as a scripting tool because of its power and flexibility as well as its availability on most platforms. However, the shell is still the scripting tool of choice for most developers, primarily because of its availability across a variety of platforms and environments.

When assessing shell scripts, check each command for the following conditions:

- The command is unavailable on Oracle Solaris 11.
- The command is in a different location and the location is not in the user's path.
- Multiple implementations of the command are available on the system (legacy, GNU, XPG4, XPG6, and so on) and PATH is picking up the right ones.
- The command uses an option that does not exist on Oracle Solaris 11.
- The command uses an option that has different functionality on Oracle Solaris 11.
- The output of the command is different and is redirected.

Build Environment Dependencies

It is very important to choose the right set of tools and build environment in order to reduce the migration effort to the barest minimum. It should be noted that almost all the open source build tools (GNU/GPL) and utilities popularly used on RHEL are also available on Oracle Solaris 11. It is also important to note that Oracle Solaris Studio 12.x and bundled utilities provide a very powerful build environment that can help you get the most out of your applications on Oracle Solaris and Oracle

systems. Oracle Solaris Studio 12.x is available on both RHEL and Oracle Solaris (SPARC as well as x64).

It is much easier to transition from RHEL to Oracle Solaris if you can maintain the same build tools and build environment on the two systems.

The following points need to be taken into consideration while finalizing the target build environment:

- Build tools and other build dependencies (`gmake`, `dmake`, `make`, ANT, and so on)
- Tools used by the applications
- Command-line options provided by the tools

Assess Food Chain Dependencies

Another very important factor that needs special attention is dependency on third-party components. For example, check whether the applications use or depend on the following:

- Any third-party proprietary libraries available in the public domain as a ready-made binary (no source code)
- Open source code or open source library
- The order in which symbols are resolved; that is, which symbols get resolved from which library if symbols with the same name are defined (implemented) in multiple libraries

The most important part of the migration process is to check for the availability of the above mentioned dependencies on the target Oracle Solaris 11 platform, because sometimes the availability of a third-party dependency can become a limiting factor.

Below are some guidelines that will not only help reduce migration effort but also help the applications work better on Oracle Solaris 11:

- Evaluate the benefit of using native tools (Oracle-provided tools) versus open source tools. Choosing the right tools and libraries and, at times, changing the environment to a native implementation can be beneficial. In almost all cases, you will find that the return on investment (ROI) and operational improvements you gain by transitioning to an Oracle Solaris 11 implementation are compelling and significant.
- Check whether you can upgrade to the latest libraries and scalable infrastructure without affecting the supported functionality of the existing applications.
- Explore the availability of Oracle Solaris built-in features, infrastructure, and tools that can provide similar functionality.
- Look for alternatives from different venders providing similar functionality.

Data Portability

Data Migration Considerations

Data migration is one of the most challenging tasks in the porting process. Data migration activity is primarily divided into two parts:

- Migration of raw data, which includes migration of application data, schema, tables, indexes, constraints, and so on.
- Migration of associated infrastructure, which includes migration of stored procedures, database triggers, SQL queries, functions, and so on.

Data migration is the process of converting data from one format to another, and it is an important component of any porting effort if data is to be readable on the target system. Data migration can involve file systems, file content, applications, and database content. Data migration becomes more challenging when the stored data is in an encoded format or it is in a format that is incompatible with the receiving system. Fortunately, RHEL and Oracle Solaris use ASCII to store textual data and a standard text file format.

Many data migration tools and toolkits are available in the market, and there are also many free or paid support services offered by database vendors for migrations. By using them, enterprises can realize significant time and cost savings during the migration and testing process.

RHEL and Oracle Solaris 11 provide many common GNU and legacy applications and utilities for managing data. For example, the GNU tape archive utility (`gtar`) uses a similar data format and provides common options in both environments. On Oracle Solaris, you will also find the Oracle Solaris legacy `tar` utility which has some different options than the utility on RHEL. For many of the commands and utilities, you will find more options available on Oracle Solaris than on RHEL. One reason is that Oracle Solaris has optional, installable packages with GNU utilities bundled in it, and another reason is that you still have access to legacy Oracle Solaris utilities that provide similar functionality with minor implementation differences. As a result, developers already familiar with the RHEL environment will be able to work seamlessly on Oracle Solaris without having to move from their favorite tools and utilities. This commonality is true for many other applications and utilities, and it can yield significant benefits during and after the data migration.

Data Portability, Well-Known Issues, and Solutions

File systems are neutral to endianness in general, and swapping files is not an issue between SPARC and x86/x64 (Intel or AMD) versions of Oracle Solaris. However, applications storing raw data that needs to be shared across platforms can become an issue.

For example, if an application on the Oracle Solaris OS for SPARC platforms writes the data structures in a raw format to the files, the data stored in these files would be endian-dependent. Reading from or writing to these same data files from an Oracle Solaris system that is based on Intel or AMD processors can create problems regarding the endianness of the data. In short, the binary (raw) data stored in a file is generally not transferable between SPARC and x86/x64 (Intel or AMD) platforms.

Applications storing data that is shared between platforms can handle the endianness issues in one of the following two ways:

- Store data in an application-defined, endian-neutral format using text files and strings.
- Choose either the big-endian or little-endian convention and do byte swapping (potentially using enabling technology such as XDR) when necessary.

The need for cross-platform compatibility is so well understood that major applications have been available on big-endian and little-endian Oracle Solaris environments for years without problems. They range from personal productivity applications to major database management systems from vendors including Oracle, Informix, and Sybase.

While there are many similarities between a database running on RHEL and one running on Oracle Solaris, simply moving a database from one platform to the other usually requires some data transformation. If the database product is available on both platforms *from the same vendor*, this task might become as simple as exporting the database running on RHEL to a standardized file format and then importing it into a new database on Oracle Solaris. When the port also involves a change in database vendors, more extensive data transformations might be required.

Most enterprise applications rely on information stored in databases to satisfy user requests. At a broader level, databases can be classified in two categories:

- Open source databases
- Proprietary databases

The database choice is driven by application needs, cost, and business needs. If the same database vendor can be maintained on both platforms, the migration process is much simpler and straightforward. Fortunately, most of the databases in both the categories are available on both Linux and Oracle Solaris.

Here is a list of the most popular open source and proprietary databases and their availability on both platforms.

TABLE 2-1. DATABASE SUPPORT

DATABASE	LATEST VERSION	RHEL	ORACLE SOLARIS
POPULAR OPEN-SOURCE DATABASES			
MySQL	5.6.x	√	√
PostgreSQL	9.1.x	√	√
SQLLite	3.7.x	√	√
Ingres	10.x	√	√
Berkeley DB	5.x	√	√

POPULAR PROPRIETARY DATABASES			
Oracle	11gR2	√	√
DB2	9.7.x	√	√
Sybase	15.7.x	√	√
Informix	11.7.x	√	√

Though it is challenging to move data between proprietary databases, most of the database vendors ship tools to aid this type of data migration.

For information about migrating from various proprietary databases to Oracle Database, please refer to http://www.oracle.com/technetwork/products/migration/index-084442.html.

Application Verification

Code Coverage

Functional testing is the most critical phase in the migration process. Many of the system calls and features on both platforms look similar, but there are subtle behavioral differences that can be uncovered only during testing. Even if the code compiles on the new platform, it might behave significantly different during actual test runs. Hence, it is very important to ensure that the applications are thoroughly tested on the target platform.

Oracle provides powerful tools to help track code coverage during testing as well as to unearth difficult to detect code issues, such as memory leaks. One such tool is Uncover, which is a code-coverage tool bundled with Oracle Solaris Studio software.

To use Uncover on Oracle Solaris 11, the following steps must be performed:

- The code must be compiled with Oracle Solaris Studio optimization flags `-xO[n]`. For example, a typical compilation command line looks like the following:

  ```
  cc -g -xO2 test.c
  ```

- Instrumentation codes should be added to the binary using the following command:

  ```
  uncover a.out
  ```

- When the instrumented binary is run and regular functional testing is performed, the code-coverage data will be stored in the following directory:

  ```
  /tmp/project/a.out.uc
  ```

- Then this Oracle Solaris Studio tool can be used to generate the report:

```
uncover a.out.uc
```

A GUI will then appear for viewing coverage data.

Memory Leak Detection

One of the most powerful tools that is bundled with Oracle Solaris Studio is Discover, which is a memory error discovery tool for detecting memory access errors and leaks.

To use Discoverer, the following steps must be performed:

- As a prerequisite, the code must be compiled with Oracle Solaris Studio compiler optimization flags `-xO[n]`:

  ```
  cc -g -xO2 test.c
  ```

- Instrumentation codes need to be added to the binary:

  ```
  discover -w text a.out
  ```

  Note: Discover automatically adds instrumentation to shared libraries when they are opened.

- Once instrumentation is added, the instrumented binary can be run and the usual testing, including stress tests, can be continued.
- The tool will track all the memory leaks during the runs, which can be viewed through the report generated by the tool.

Chapter 3 Operating System Considerations

Supported Processor Architectures (Linux Versus Oracle Solaris)

Many factors have resulted in increased use of Linux systems during last decade; in particular, significant popularity was observed in desktop systems and to some extent in the server systems market as well. One of the major reasons for this increase in deployments was the desire to decrease system costs. Though various flavors of Linux are available on SPARC (RISC) platforms, most Linux deployments happened on x86/x64 systems. On the other hand, most enterprise deployments continue to happen on RISC platforms due to their inherent reliability, stability, availability, and scalability.

Oracle Solaris 11 is designed to address both of these markets. The availability of Oracle Solaris on x86/x64 and SPARC platforms, along with the source code compatibility guarantee across processor architectures offered by Oracle, makes development on Oracle Solaris a compelling proposition.

Migrating from Linux to Oracle Solaris while remaining on an x86/x64 system is a simple task because few processor-specific issues arise during this transition. On the other hand, specific measures must be taken when transitioning to a SPARC/RISC platform.

The next few sections discuss in detail the various aspects that must be considered during migration from an x86/x64 system to an Oracle Solaris SPARC platform.

Storage Order and Alignment

Default Alignment and Recommendations

Every data type has alignment requirements mandated by the processor architecture. A processor will be able to efficiently process data if the processing word size matches processor's data bus size. For example, on a 32-bit machine, the processing word size is 4 bytes.

The reason for not permitting misaligned long-word reads and writes is obvious. For example, an aligned integer A would be written as A0, A1, A2, and A3 in the memory (4 bytes). If this integer is stored with proper alignment, the processor can read the complete word in a single bus cycle.

Figure 3-1. A properly aligned integer value can be read in a single fetch operation.

If the same processor now attempts to access an integer at an unaligned address, it will have to read bytes X0, X1, X2, and X3 (see Figure 3.2). This read cannot be performed in a single 32-bit bus cycle.

The processor will have to issue two different read instructions to read the complete integer. Thus, it takes twice as long to read a misaligned integer. In short, the addresses of memory chunks should be in multiples of their sizes. If an address satisfies this requirement, it is said to be properly aligned. The consequences of accessing data via an unaligned address can range from slower execution to program termination.

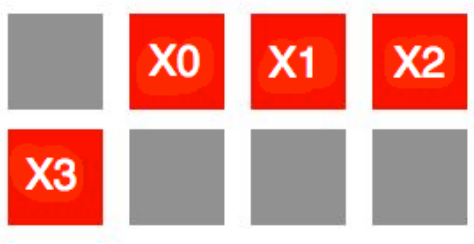

Figure 3-2. An improperly aligned integer value would require two fetch operations.

Data Structures and Sizes

In order to generate efficient code, compilers have to follow the byte alignment restrictions defined by the target processors. This means that compilers have to add pad bytes into user-defined structures so that the structure does not violate any restrictions imposed by the target processor. The compiler padding is illustrated in the following example. Here, an `int` is assumed to be 4 bytes, a `short` is 2 bytes, and a `char` is a single byte.

```
struct mydata {
char C;
long L;
short B;
long J;
};
```

Since the alignment of an `int` on this platform is 4 bytes, 3 bytes are added after `char c`, and two bytes are added at the end of `short b`, as shown in Figure 3-3. The size of this `struct` is 16 bytes on an x86 system and 32 bytes on a SPARC (64-bit) system. By padding, the address of this `struct` data is divisible evenly by 4. This is called structure member alignment. Of course, the size of `struct` has grown as a consequence.

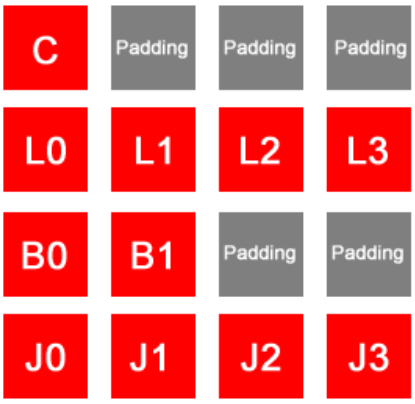

Figure 3-3. Visual representation of how `struct mydata` would be padded to align with 4-byte boundaries.

Compiler Options and Portability of Code

The `pack` pragma directive can be used to specify different packing alignment for structure, union, or class members.

```
#pragma pack(push, 1)
struct mystruct
{
char c1;    // 1-byte
double d2;  // 8-byte
};
#pragma pack(pop)
```

Most compilers provide nonstandard extensions (for example, pragmas or command-line switches) to switch off the default padding. Consult the documentation provided by the compiler for more details. Be aware of using custom structure member alignment, because this can cause serious compatibility issues, for example, when you pass a custom-aligned structure to a function from an external library that is using different packing alignments. To avoid such problems, it is almost always better to use default alignment.

In some cases, it is mandatory to avoid padded bytes among the members of a structure. For example, take the case of an application whose primary job is to send serialized data over a network. Avoiding byte padding can drastically improve the network utilization.

However, care should be exercised in accessing structure members at the other end. Typically, reading byte-by-byte is an option for avoiding misalignment errors.

It should be clear by now that to be able to transfer the raw data from one platform and load it on another, the two platforms not only need to have *fundamental types* of the same size and of the same endianess, but they also need to be *alignment-compatible*. Otherwise, the positions of members inside the

type—and even the size of the type itself—can differ. And this is exactly what happens if the data corresponding to `mystruct` is moved between an x86/x64 system and a SPARC system, even though the types used in the `struct` are of the same size.

Byte Ordering

Different microprocessor vendors use different byte-ordering schemes. For example, Intel processors have traditionally been little-endian. Motorola processors have always been big-endian. Big-endian is an order in which the "big end" (the most-significant byte) is stored first. Little-endian is an order in which the "little end" (the least-significant byte) is stored first.

Figure 3-4 and Figure 3-5 show a representation of the hexadecimal number 0xFF342109 on a big-endian and little-endian machine. The contents of memory locations 0x1000 to 0x1003 are shown.

Figure 3-4. Representation of 0xFF342109 on a little-endian system such as with Linux on an x86 system.

Figure 3-5. Representation of 0xFF342109 on a big-endian system such as with Oracle Solaris on a SPARC system.

Data Conversion for Interoperability

The logic for converting between the big-endian and little-endian formats is actually quite straight forward. Looking at the diagrams above, it is obvious that just by swapping the appropriate bytes, one format can be converted to the other.

To account for communication over a network that has machines with diverse architectures, generally the data is converted in “network byte order” at the transmitting end before being sent to the network, and the received data is converted back to “host byte order” after the receipt of the packet at the destination host. Ready-made library conversion routines are shipped with the operating system to help with this conversion, for example:

```
ntohl( ) // Network-to-host byte order
htonl() // Host–to-network byte order
```

The story is little different for member data in a structure, union, or class objects. The `struct` member variables must be aligned to the highest bytes of the size of any member variables to prevent performance penalties.

```
// 4-byte alignment
struct mystruct
{
    char a; // size = 1 byte
                // 3 bytes padding
    int i;    // size = 4 bytes
};
```

In the above example, the size of `mystruct` is 8 bytes.

```
// 8-byte alignment
struct mystruct_1
{
    char a;         // size = 1 byte
                        // 7 bytes padding
    double i;     // size = 8 bytes
};
```

In the above example, the size of `mystruct_1` is 16 bytes.

Low-Level Code, Bit-Level Operations

When migrating from a 32-bit RHEL application to a 64-bit Oracle Solaris SPARC application, the bit-shifting operations used inattentively in typical legacy C code can cause lots of problems. The following example is a function that does a shift operation. Note that untyped integral constants are of type `unsigned int`. During shifting, this can lead to unexpected truncation.

For example, in the following code, the maximum value for `i` can be 31. This is because the type of "`1 << i`" is `int`.

```
long j = 1 << i;
```

To get the shift done on a 64-bit SPARC system, `1L` should be used, as shown below:

```
long j = 1L << i;
```

System APIs

System Call Mapping

Both RHEL and Oracle Solaris are UNIX operating systems that follow the POSIX standard. Both provide well-defined system call interfaces. Most of the system calls available on RHEL are also

available on Oracle Solaris, either as system calls or library functions (APIs). There are some minor differences in system call implementation on the two platforms.

Table 3-1 lists the implementation differences in terms of differences in the number of parameters, return values, or changes in the headers file names where the system call signatures are declared. The table also gives the workaround or alternate function call that needs to be used on Oracle Solaris 11 so that code originally written to work on RHEL can compile and work on Oracle Solaris 11.

There are also some system calls that are available on only one platform. For such system calls, more time and effort will have to be spent during the migration.

TABLE 3-1. EQUIVALENT SYSTEM CALLS, ALTERNATIVE SYSTEM CALLS, AND WORKAROUNDS

API	SYNOPSIS ON LINUX	FOR ORACLE SOLARIS 11
access	#include <unistd.h> int access(const char *pathname, int mode);	Signature on Oracle Solaris is the same. On Oracle Solaris, need to additionally include: #include <sys/fcntl.h>
chown	#include <unistd.h> int chown(const char *path, uid_t owner, gid_t group);	Signature on Oracle Solaris is the same. On Oracle Solaris, need to additionally include:#include <sys/types.h>
creat	#include <sys/types.h> #include <sys/stat.h> #include <fcntl.h> int creat(const char *pathname, mode_t mode);	Signature on Oracle Solaris is the same. Does not require sys/types.h.
faccessat	#define _ATFILE_SOURCE #include <fcntl.h> /* Definition of AT_* constants */ #include <unistd.h> int faccessat(int dirfd, const char *pathname, int mode, int flags);	Signature on Oracle Solaris is the same. Requires #include <sys/fcntl.h> instead of #include <fcntl.h>. #define _ATFILE_SOURCE not required.
fchmodat	#define _ATFILE_SOURCE #include <fcntl.h> /* Definition of AT_* constants */ #include <sys/stat.h> int fchmodat(int dirfd, const char *pathname, mode_t mode, int flags);	Signature on Oracle Solaris is the same. Does not require #define _ATFILE_SOURCE and #include <fcntl.h>.
fchown	#include <unistd.h> int fchown(int fd, uid_t owner, gid_t group);	Signature on Oracle Solaris is the same. Requires #include <sys/types.h>.
fchownat	#define _ATFILE_SOURCE #include <fcntl.h> /* Definition of	Signature on Oracle Solaris is the same. Does not require #define _ATFILE_SOURCE and

TABLE 3-1. EQUIVALENT SYSTEM CALLS, ALTERNATIVE SYSTEM CALLS, AND WORKAROUNDS

API	SYNOPSIS ON LINUX	FOR ORACLE SOLARIS 11
	AT_* constants */ #include <unistd.h> int fchmodat(int dirfd, const char *pathname, mode_t mode, gid_t group, int flags);	#include <fcntl.h>. Requires #include <sys/types.h>.
fcntl	#include <unistd.h> #include <fcntl.h> int fcntl(int fd, int cmd, ... /* arg */);	Signature on Oracle Solaris is the same. Requires #include <sys/types.h>.
fork	#include <unistd.h> pid_t fork(void);	Signature on Oracle Solaris is the same. Requires #include <sys/types.h>.
fstat	#include <sys/types.h> #include <sys/stat.h> #include <unistd.h> int fstat(int fd, struct stat *buf);	Signature on Oracle Solaris is the same. Requires #include <fcntl.h>. Does not require #include <unistd.h>.
futimesat	#define _ATFILE_SOURCE #include <fcntl.h> /* Definition of AT_* constants */ int futimesat(int dirfd, const chat *pathname, const struct timeval times[2]);	Signature on Oracle Solaris is the same. Does not require #define _ATFILE_SOURCE and #include <fcntl.h>. Requires #include <sys/time.h>.
getdents	int getdents(unsigned int fd, struct linux_dirent *dirp,unsigned int count);	int getdents(int fildes, struct dirent *buf, size_t nbyte); Requires #include <dirent.h>.
getgroups	#include <sys/types.h> #include <unistd.h> #include <grp.h> int getgroups(int size, gid_t list[]);	int getgroups(int gidsetsize, gid_t *grouplist); Does not require #include <sys/types.h>.
getpid	#include <sys/types.h> #include <unistd.h> pid_t getpid(void);	Signature on Oracle Solaris is the same. Does not require #include <sys/types.h>.
getppid	#include <sys/types.h> #include <unistd.h> pid_t getppid(void);	Signature on Oracle Solaris is the same. Does not require #include <sys/types.h>.
getrlimit	#include <sys/time.h> #include <sys/resource.h> int getrlimit(int resource, struct rlimit *rlim);	Signature on Oracle Solaris is the same. Does not require #include <sys/time.h>.
ioctl	#include <sys/ioctl.h>	Signature on Oracle Solaris is the same.

TABLE 3-1. EQUIVALENT SYSTEM CALLS, ALTERNATIVE SYSTEM CALLS, AND WORKAROUNDS

API	SYNOPSIS ON LINUX	FOR ORACLE SOLARIS 11
	`int ioctl(int d, int request, ...);`	Requires `#include <unistd.h>` and `#include <stropts.h>`. Does not require `#include <sys/ioctl.h>`.
`lchown`	`#include <unistd.h>` `int lchown(const char *path, uid_t owner, gid_t group);`	Signature on Oracle Solaris is the same. Requires `#include <sys/types.h>`.

`linkat`	`#define _ATFILE_SOURCE`
	`#include <fcntl.h> /* Definition of AT_* constants */ #include <unistd.h>`
	`int linkat(int olddirfd, const char *oldpath, int newdirfd, const char *newpath, int flags);`
		Signature on Oracle Solaris is the same.
		Does not require `#define _ATFILE_SOURCE` and `#include <fcntl.h>`.

`lstat`	`#include <sys/types.h>`
	`#include <sys/stat.h>`
	`#include <unistd.h>`
	`int lstat(const char *path, struct stat *buf);`
		`int stat(const char *restrict path, struct stat *restrict buf);`
		Requires `#include <fcntl.h>`.
		Does not require `#include <unistd.h>`.

`mincore`	`#include <unistd.h>`
	`#include <sys/mman.h>`
	`int mincore(void *addr, size_t length, unsigned char *vec);`
		`int mincore(caddr_t addr, size_t len, char *vec);`
		Requires `#include <sys/types.h>`.
		Does not require `#include <unistd.h>` and `#include <sys/mman.h>`.

`mkdir`	`#include <sys/stat.h>`
	`#include <sys/types.h>`
	`int mkdir(const char *pathname, mode_t mode);`
		Signature on Oracle Solaris is the same.
		Does not require `#include <sys/types.h>`.

`mkdirat`	`#define _ATFILE_SOURCE`
	`#include <fcntl.h> /* Definition of AT_* constants */ #include <sys/stat.h>`
	`int mkdirat(int dirfd, const char *pathname, mode_t mode);`
		Signature on Oracle Solaris is the same.
		Does not require `#define _ATFILE_SOURCE` and `#include <fcntl.h>`.

`mknod`	`#include <sys/types.h>`
	`#include <sys/stat.h>`
	`#include <fcntl.h>`
	`#include <unistd.h>`
	`int mknod(const char *pathname, mode_t mode, dev_t dev);`
		Signature on Oracle Solaris is the same.
		Does not require `#include <sys/types.h>`, `#include <fcntl.h>`, and `#include <unistd.h>`.

`mknodat`	`#define _ATFILE_SOURCE`	Signature on Oracle Solaris is the same.

TABLE 3-1. EQUIVALENT SYSTEM CALLS, ALTERNATIVE SYSTEM CALLS, AND WORKAROUNDS

API	SYNOPSIS ON LINUX	FOR ORACLE SOLARIS 11
	`#include <fcntl.h> /* Definition of AT_* constants */ #include <sys/stat.h>` `int mknodat(int dirfd, const char *pathname, mode_t mode, dev_t dev);`	Does not require `#define _ATFILE_SOURCE` and `#include <fcntl.h>`.
`mount`	`#include <sys/mount.h>` `int mount(const char *source, const char *target, const char *filesystemtype, unsigned long mountflags,const void *data);`	`int mount(const char *spec, const char *dir, int mflag, char *fstype, char *dataptr,int datalen, char *optptr,int optlen);` Requires `#include <sys/types.h>` and `#include <sys/mntent.h>`.
`mprotect`	`#include <sys/mman.h>` `int mprotect(const void *addr, size_t len, int prot);`	`int mprotect(void *addr, size_t len, int prot);`

`msgctl`	`#include <sys/types.h>`
	`#include <sys/ipc.h>`
	`#include <sys/msg.h>`
	`int msgctl(int msqid, int cmd, struct msqid_ds *buf);`
		Signature on Oracle Solaris is the same.
		Does not require `#include <sys/types.h>` and `<sys/ipc.h>`.

`msgget`	`#include <sys/types.h>`
	`#include <sys/ipc.h>`
	`#include <sys/msg.h>`
	`int msgget(key_t key, int msgflg);`
		Signature on Oracle Solaris is the same.
		Does not require `#include <sys/types.h>` and `#include <sys/ipc.h>`.

`msgrcv`	`#include <sys/types.h>`
	`#include <sys/ipc.h>`
	`#include <sys/msg.h>`
	`ssize_t msgrcv(int msqid, void *msgp, size_t msgsz, long msgtyp,int msgflg);`
		`ssize_t msgrcv(int msqid, void *msgp, size_t msgsz,long int msgtyp, int msgflg);`
		Does not require `#include <sys/types.h>` and `<sys/ipc.h>`.

`msgsnd`	`#include <sys/types.h>`
	`#include <sys/ipc.h>`
	`#include <sys/msg.h>`
	`int msgsnd(int msqid, const void *msgp, size_t msgsz, int msgflg);`
		Signature on Oracle Solaris is the same.
		Does not require `#include <sys/types.h>` and `<sys/ipc.h>`.

`ppoll`	`#define _GNU_SOURCE`
	`#include <poll.h>`
	`#include <poll.h>`
	`int ppoll(struct pollfd *fds, nfds_t nfds,const struct timespec *timeout, const sigset_t *sigmask);`
		`int ppoll(struct pollfd *restrict fds, nfds_t nfds,const struct timespec *restrict timeout,const sigset_t *restrict sigmask);`
		Does not require `#define _GNU_SOURCE`.

TABLE 3-1. EQUIVALENT SYSTEM CALLS, ALTERNATIVE SYSTEM CALLS, AND WORKAROUNDS

API	SYNOPSIS ON LINUX	FOR ORACLE SOLARIS 11
profil	#include <unistd.h> int profil(unsigned short *buf, size_t bufsiz, size_t offset, unsigned, int scale);	void profil(unsigned short *buff, unsigned int bufsiz, unsigned int offset,unsigned int scale);
readlink	#include <unistd.h> ssize_t readlink(const char *path, char *buf, size_t bufsiz);	ssize_t readlink(const char *restrict path, char *restrict buf, size_t bufsiz);
readlinkat	#define _ATFILE_SOURCE #include <fcntl.h> /* Definition of AT_* constants */ #include <unistd.h> int readlinkat(int dirfd, const char *pathname, char *buf, size_t bufsiz);	ssize_t readlinkat(int fd, const char *restrict path, char *restrict buf, size_t bufsize); Does not require #define _ATFILE_SOURCE and #include <fcntl.h>.
renameat	#define _ATFILE_SOURCE #include <fcntl.h> /* Definition of AT_* constants */ #include <stdio.h> int renameat(int olddirfd, const char *oldpath, int newdirfd, const char *newpath);	Signature on Oracle Solaris is the same. Requires only #include <unistd.h>.
semop	#include <sys/types.h> #include <sys/ipc.h> #include <sys/sem.h> int semop(int semid, struct sembuf *sops, unsigned nsops);	int semop(int semid, struct sembuf *sops, size_t nsops);
semtimedop	#include <sys/types.h> #include <sys/ipc.h> #include <sys/sem.h> int semtimedop(int semid, struct sembuf *sops, unsigned nsops,struct timespec *timeout);	int semtimedop(int semid, struct sembuf *sops, size_t nsops,const struct timespec *timeout);
setgroups	#include <grp.h> int setgroups(size_t size, const gid_t *list);	int setgroups(int ngroups, const gid_t *grouplist); Requires <unistd.h> and does not require <grp.h>.
setpgid	#include <unistd.h> int setpgid(pid_t pid, pid_t pgid);	Signature on Oracle Solaris is the same. Requires #include <sys/types.h>.
setsid	#include <unistd.h> pid_t setsid(void);	Signature on Oracle Solaris is the same. Requires #include <sys/types.h>.
shmctl	#include <sys/ipc.h> #include <sys/shm.h>	Signature on Oracle Solaris is the same. Also requires #include <sys/types.h>.

TABLE 3-1. EQUIVALENT SYSTEM CALLS, ALTERNATIVE SYSTEM CALLS, AND WORKAROUNDS

API	SYNOPSIS ON LINUX	FOR ORACLE SOLARIS 11
	`int shmctl(int shmid, int cmd, struct shmid_ds *buf);`	

shmget	`#include <sys/ipc.h>`
	`#include <sys/shm.h>`
	`int shmget(key_t key, size_t size, int shmflg);`
		Signature on Oracle Solaris is the same.
		Also requires `#include <sys/types.h>`.

sigaction	`#include <signal.h>` `int sigaction(int signum, const struct sigaction *act, struct sigaction *oldact);`	`int sigaction(int sig, const struct sigaction *restrict act,struct sigaction *restrict oact);`
sigaltstack	`#include <signal.h>` `int sigaltstack(const stack_t *ss, stack_t *oss);`	`int sigaltstack(const stack_t *restrict ss, stack_t *restrict oss);`
sigprocmask	`#include <signal.h>` `int sigprocmask(int how, const sigset_t *set, sigset_t *oldset);`	`int sigprocmask(int how, const sigset_t *restrict set,sigset_t *restrict oset);`

stat	`#include <sys/types.h>`
	`#include <sys/stat.h>`
	`#include <unistd.h>`
	`int stat(const char *path, struct stat *buf);`
		`int stat(const char *restrict path, struct stat *restrict buf);`
		Requires `#include <fcntl.h>`.
		Does not require `#include <unistd.h>`.

stime	`#include <time.h>` `int stime(time_t *t);`	`int stime(const time_t *tp);` Requires `#include <unistd.h>`. Does not require `#include <time.h>`.

symlinkat	`#define _ATFILE_SOURCE`
	`#include <fcntl.h> /* Definition of AT_* constants */ #include <stdio.h>`
	`int symlinkat(const char *oldpath, int newdirfd, const char *newpath);`
		Signature on Oracle Solaris is the same.
		Requires `#include <unistd.h>`.
		Does not require `#define _ATFILE_SOURCE`, `#include <fcntl.h>`, and `#include <stdio.h>`.

sysfs	`int sysfs(int option, const char *fsname);` `int sysfs(int option, unsigned int fs_index, char *buf); int sysfs(int option);`	`int sysfs(int opcode, int fs_index, char *buf);` Requires `#include <sys/fstyp.h>` and `#include <sys/fsid.h>`.
sysinfo	`#include <sys/sysinfo.h>` `int sysinfo(struct sysinfo *info);`	`int sysinfo(int command, char *buf, long count);` Requires `#include <sys/systeminfo.h>`. Does not require `#include <sys/sysinfo.h>`.

TABLE 3-1. EQUIVALENT SYSTEM CALLS, ALTERNATIVE SYSTEM CALLS, AND WORKAROUNDS

API	SYNOPSIS ON LINUX	FOR ORACLE SOLARIS 11
time	#include <time.h> time_t time(time_t *t);	Signature on Oracle Solaris is the same. Requires #include <sys/types.h>.
times	#include <sys/times.h> clock_t times(struct tms *buf);	Signature on Oracle Solaris is the same. Requires #include <limits.h>.
unlinkat	#define _ATFILE_SOURCE #include <fcntl.h> int unlinkat(int dirfd, const char *pathname, int flags);	Signature on Oracle Solaris is the same. Requires only #include <unistd.h>.
ustat	#include <sys/types.h> #include <unistd.h> /* libc[45] */ #include <ustat.h> /* glibc2 */ int ustat(dev_t dev, struct ustat *ubuf);	Signature on Oracle Solaris is the same. Does not require #include <unistd.h>.
vhangup	#include <unistd.h> int vhangup(void);	void vhangup(void);

Chapter 4 Application Development Environment

Reducing Migration Costs

Oracle Solaris 11 provides a very large set of native (Oracle proprietary) and open source commands, tools, libraries, and platform services similar to those available on RHEL. Therefore, the migration of an application development environment from Linux to Oracle Solaris can be very smooth. Oracle Solaris 11 provides enables using open source tools and libraries alongside the native infrastructure. For example, NetBeans can be configured to use GCC as the default compiler.

If the same development tools and compilers are used on both sides, the tool-related porting complexities become minimal. Oracle has made Oracle Solaris Studio 12.x compilers and tools available on both the Oracle Solaris and RHEL platforms. It is also important to note that the GNU compiler collection GCC and other GNU tools are available on both RHEL and Oracle Solaris 11.

A first step towards migration to Oracle Solaris can be moving to Oracle Solaris Studio development on the Linux platform.

The benefit of using native tools (Oracle-provided tools) instead of open source tools is enormous. At times, changing the environment to a native implementation provides more value compared to the effort required for migration. In almost all the cases, the return on investment (ROI) and operational improvements reaped by transitioning to an Oracle Solaris 11 implementation are very compelling and significant; hence, it is strongly recommended to plan for migrating to an Oracle-provided, native build infrastructure (Oracle Solaris Studio tools and libraries) once the initial porting is completed.

Oracle Solaris Studio

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

In particular, Oracle Solaris Studio tools are designed to leverage the capabilities of multicore CPUs including Oracle's SPARC T4, SPARC T3, SPARC T2, and SPARC Enterprise M-Series processors, as well as the Intel Xeon and AMD Opteron processors. The tools enable easier creation of parallel and concurrent software applications for these platforms. The compilers, tools, and libraries shipped with Oracle Solaris Studio are engineered to make applications run optimally on Oracle platforms.

Oracle Solaris Studio Components

The components of Oracle Solaris Studio include the following:

- An IDE for application development in a graphical environment. The Oracle Solaris Studio IDE integrates several other Oracle Solaris Studio tools and uses Oracle Solaris technologies such as DTrace.

- C, C++, and Fortran compilers for compiling code at the command line or through the IDE. The compilers are engineered to work well with the Oracle Solaris Studio debugger (`dbx`) and include the ability to optimize code by specifying compiler options.
- Libraries to add advanced performance and multithreading capabilities to applications.
- The `dmake` utility for building code in distributed computing environments at the command line or through the IDE.
- The `dbx` bebugger for finding bugs in code at the command line, through the IDE, or through an independent graphical interface (`dbxtool`).
- Performance tools that employ Oracle Solaris technologies such as DTrace, which can be used at the command line or through independent graphical interfaces to find trouble spots in code that might be diffiuclt to detect by debugging with `dbx`.

Together, all these tools enable building, debugging, and tuning applications for high performance on Oracle Solaris running on Oracle's Sun systems.

Standards Adherence

If the application to be ported has strict requirements for adherence to standards, Oracle Solaris Studio is the way to go. The native C compiler (`cc`) available in Oracle Solaris Studio is in full compliance with the ISO/IEC 9899:1999 standard, while the C++ compiler (`CC`) supports the ISO International Standard for C++ (ISO IS 14882:2003). The Fortran compiler (`f95`) conforms to part one of the ISO/IEC 1539-1:1997 Fortran standards document. This compiler also provides a Fortran 77 compatibility mode that accepts most legacy Fortran 77 source code. The Oracle Solaris Studio compilers support OpenMP, IEEE floating point, and C99, and they adhere to IEEE 754.

If the application to be ported does *not* have strict requirements for adherence to standards, additional performance gains can be achieved by using optimization flags that enable building higher-performing binaries. For example, the C compiler option `-fns` allows nonstandard floating-point truncation.

The `-fast` macro available in Oracle Solaris Studio is the easiest way to generate an optimized binary for specific targeted hardware. (A word of caution: The `-fast` macro implies `–fns`.)

Useful Tools

Many useful tools are available on Oracle Solaris 11 that not only help generate high-performing binaries but also help isolate difficult-to-detect code issues, for example, the following Oracle Solaris Studio tools:

Performance Analyzer can be used not only to analyze application performance but also to determine what parts of a program are candidates for improvement and to help identify performance hotspots.

- **Thread Analyzer** can be used to detect difficult-to-detect code issues in multithreaded programs. For example, it helps detect data races and deadlock conditions.
- **D-light** is a plug-in for Oracle Solaris Studio 12.x, which offers instrumentation that takes advantage of the DTrace debugging and performance analysis functionality in Oracle Solaris.

- **Discover** is a tool used by software developers to detect programming errors related to the allocation and use of program memory at runtime.
- It can detect the following types of programming errors:
  - Reading from and writing to unallocated memory
  - Freeing the wrong memory blocks
  - Using freed memory
  - Accessing memory beyond allocated array bounds
  - Memory leaks
  - Accessing uninitialized memory
- **Uncover** is a command-line tool for measuring the code coverage of user applications. This tool can display information on the areas of an application that are exercised during testing. The coverage information reported by this tool could be at the level of a function, statement, basic block, or instruction. With Oracle Solaris 11, there is no longer a need to buy expensive commercial third-party memory usage and leak analyzers. By default, the high-performance `libumem` library is bundled with Oracle Solaris 11. The `libumem` library, along with debugging tool `dbx`, finds difficult-to-detect memory leaks and buffer overruns.

GNU Compiler Collection (GCC) Versus Oracle Solaris Studio Compiler Option Mapping

By default, RHEL 6 comes with GCC v 4.4.5, while on Oracle Solaris 11, the certified IPS package for GCC v 4.5.2 is available for installation. For various reasons discussed earlier, it is strongly recommend that Oracle Solaris Studio be used for all development work on both platforms.

To help with this compiler transition, Table 4-1 provides the mapping of various GCC options with Oracle Solaris Studio compiler options. Also see Appendix B, which provides a list of all GCC options that exactly match the options in Oracle Solaris Studio 12.3. The list in Appendix B includes compiler options available in Oracle Solaris Studio 12.3 that are not shown in Table 4-1.

TABLE 4-1. EQUIVALENT COMPILER FLAGS AND WORKAROUNDS

GCC 4.4.5 OPTIONS	ORACLE SOLARIS STUDIO 12.3 OPTIONS	DESCRIPTION
`-###`	`-xdryrun` or `-###`	Shows each component as it would be invoked, but does not actually execute it. Also shows how command options would expand.
`-aux-info filename`	`-xP`	Prints prototypes for K&R function definitions.
`-Bprefix` `GCC_EXEC_PREFIX`	`-YA, dir` `-Yc, dir`	`-YA` changes the default directory that is searched for the compiler components.

TABLE 4-1. EQUIVALENT COMPILER FLAGS AND WORKAROUNDS

GCC 4.4.5 OPTIONS	ORACLE SOLARIS STUDIO 12.3 OPTIONS	DESCRIPTION
(environment variable)		`-Yc` specifies that the component *c* of the compiler can be found in directory *dir*. For GCC, the `-B` option specifies the directory but not the component.
`-B-Idir`	`-YI, dir`	Changes the default directory that is searched for include files.
`-B -Ldir`	`-YP, dir`	Changes the default directory for finding library files.
`-march=cputype`	`-xchip[ =name]`	Selects the target processor.

`-mcpu=i386`
or
`-march=i386`
	`-x386`	Optimizes for the 386 processor.

`-mcpu=i486`
or
`-march=i486`
	`-x486`	Similar to `-x386`, but for the i486 processor.

`-b machine -march=cputype`	`-native` `-xtarget=native`	Directs the compiler to generate code for the current system architecture.
`-fPIC`	`-xcode`	Emits position-independent code. In Oracle Solaris Studio, `-KPIC` is obsolete. Use `-xcode` instead.
`-fsyntax-only`	`-xe`	Performs syntax checks only.
`-ftls-model=model`	`-xthreadvar[ =o]`	Controls implementation of thread local variables.
`-fno-inline`	`-xinline=no%function_name`	Does not try to inline functions.
`-finline-functions`	`-xinline=%auto`	Attempts to inline all functions.
`-floopoptimize`	`-xdepend`	Analyzes loops for inter-iteration data dependencies. GCC `-floopoptimize` performs loop optimizations.
`-fno-builtin` (default behavior is to optimize)	`-xbuiltin`	Improves optimization of code that calls standard library functions.

`-ffixed-reg`
`-fcall-used-reg`
`-fcall-saved-reg`
	`-xregs=r[ ,r. . .]`	Specifies the usage of registers for the generated code.

`-ffast-math`	`-fsimple=[ n]`	Allows the optimizer to make simplifying assumptions about floating-point arithmetic.

TABLE 4-1. EQUIVALENT COMPILER FLAGS AND WORKAROUNDS

GCC 4.4.5 OPTIONS	ORACLE SOLARIS STUDIO 12.3 OPTIONS	DESCRIPTION
`-ffloat-store`	`-[ no] fstore`	Forces conversion of floating-point expressions instead of leaving them in the register.
`-fargument-(no)alias` `-fargument -(no)alias-global`	`-xrestrict[ =f]`	Treats pointer-valued function parameters as restricted pointers.

`-fstrict-aliasing`
`-fargument-[no]alias`
`-fargument-noalias-global`
	`-xalias_level[ =l]`	Specifies what assumptions can be made to perform optimizations using type-based alias analysis.

`- f[no]unsigned-char`	`-xchar[ =o]`	Specifies whether type `char` is signed or unsigned.
`-fstack-check`	`-xcheck[=o]`	Turns on runtime checking for stack overflow.
`-fprofile-generate`	`-xprofile=collect`	Instructs the compiler to generate code for collecting profiling information.
`-fprofile-use`	`-xprofile=use`	Uses the information from various runs of a program compiled with `-xprofile=coll ect`.
`-fprofile-arcs`	`-xprofile=tcov`	Causes the program to emit information that can be examined using the `tcov` tool. For GCC, use `gcov`.
`-ftime-report`	`-xtime`	Reports the time and resources used for the compilation.
`--help`	`-xhelp=flags`	Displays a summary of the compiler options.
`-malign-double`	`-dalign` is obsolete. Use `-xmemalign=8s` instead.	Assumes at most 8-byte alignment.
`-mhard-float`	`-fma[=none|fused]`	Enables generation of floating point, fused, and multiply-add instructions.
`-M`	`-xM`	Generates makefile dependencies.
`-MD`	`-xMD`	Generates makefile dependencies including compilation.
`-MF`	`-xMF`	Specifies a file name for the makefile dependency output.
`-MM`	`-xM1`	Same as `-M`, but excludes files in `/usr/include`. GCC excludes files from system header directories.
`-MMD`	`-xMMD`	Generates makefile dependencies excluding system header

TABLE 4-1. EQUIVALENT COMPILER FLAGS AND WORKAROUNDS

GCC 4.4.5 OPTIONS	ORACLE SOLARIS STUDIO 12.3 OPTIONS	DESCRIPTION
		files.
`-nostdlib`	`-xnolib`	Does not automatically link in any libraries.
`-Os`	`-xspace`	Performs only optimizations that do not increase the code size.
`-pg`	`-xpg`	Prepares the code to collect data for profiling with `gprof`.
`-pthread`	`-mt`	Passes `-D_REENTRANT` to the preprocessor and adds the threads library to the link line.
`-save-temps`	`-keeptmp`	Retains temporary files.
`-shared`	`-G`	Causes the linker to create a shared object instead of an executable. (Also use `-Kpic` with GCC and `-xcode` with Oracle Solaris Studio.)
`-static`	`-d[ y|n]`	Specifies dynamic or static linking.
`-std={ c99|c89}`	`-xc99[=`*o*`]`	Controls recognition of features from the C99 standard.
`TMPDIR` (environment variable)	`-xtemp=`*dir*	Sets the directory for temporary files. With GCC, set the environment variable `TMPDIR` to the name of the directory you want to use.
`-trigraphs`	`-xtrigraphs`	Determines whether the compiler recognizes trigraph sequences as defined by the ISO C standard.
`-v`	`-V`	Prints information about the version of each tool as the compiler executes.
`--version`	`-xlicinfo` (obsolete)	Returns version, copyright, and license information about the compiler.
`-w`	`-erroff[=`*t*`]`	Disables printing specific warnings based on their tag number. In GCC, `-w` inhibits all warnings.
`-Wall`	`-erroff`	Causes the compiler to perform more semantic checks and enables lint-like tests. This can be achieved with GCC by using the `-Wall` option and other `-W` options that are not included in `-Wall`.
`-Werror`	`-errwarn=t`	Treats warnings as errors.
`-Wstrict-prototypes`	`-fd`	Warns about K&R-style function declarations and definitions.
`-std=iso*`	`-X[ a | c | s | t]`	Selects various degrees of compliance with ISO C.
`-funroll-loops -funroll-all-loops`	`-xunroll=`*n*	Instructs the compiler to unroll loops.

TABLE 4-1. EQUIVALENT COMPILER FLAGS AND WORKAROUNDS

GCC 4.4.5 OPTIONS	ORACLE SOLARIS STUDIO 12.3 OPTIONS	DESCRIPTION
`-ffast-math`	`-xlibmopt`	Enables the compiler to use a library of optimized math routines.

Dynamic Linking Using Hardware-Specific Performance Libraries

Taking advantage of hardware properties specific to target platforms can result in a significant performance improvement. For example, numerically intensive applications can utilize the specific features of a target SPARC platform, such as the number of registers available for computation, memory latency, and so on while generating the binaries for independent software vendor (ISV) applications. The CPU-specific instruction set or the amount of data prefetching that is advantageous on the target hardware architecture can be used, but note that this might not work or it might degrade the performance on the other processors. It is not always feasible for an ISV to develop and distribute different versions of an application specifically tuned for different platforms. However, some features of the Oracle Solaris runtime linker allow different optimized libraries to be used on different architectures within a single version of the application (transparent to the user).

One way to build platform-specific libraries is to use the `$PLATFORM` linker token, which can be specified as a part of the runtime library path. The `$PLATFORM` token will expand at runtime as the output of the `uname -i` command. As an example, consider two implementations of the same library: one optimized for platforms based on the sun4u processor architecture and the default version for all other Sun machines from Oracle:

```
$ find /opt/ISV -name libtmp.so
/opt/ISV/lib/sun4u/libtmp.so
/opt/ISV/lib/other/libtmp.so
```

Linking these libraries with the `$PLATFORM` token results in an executable that resolves the correct library depending on the architecture detected at runtime.

```
$ setenv LD_OPTIONS '-R /opt/ISV/lib/$PLATFORM:/opt/ISV/lib/other'
```

Note: The `/opt/ISV/lib/other` directory explicitly specified in the runtime linker path ensures that the default version of the `/opt/ISV/lib/other/libtmp.so` library will be used on machines running versions of Oracle Solaris earlier than 2.6; anything else is, therefore, not capable of using the `$PLATFORM` functionality.

In addition to `$PLATFORM`, tokens `$OSNAME` and `$OSREL` can be used, which expand to the `uname -s` and `uname -r` outputs, respectively. The Oracle Solaris linker also allows shared libraries that are specific to a particular instruction set to be specified via the `$ISALIST` token. At runtime, this token is replaced by each of the native instruction sets on the platform shown by the `isalist` command until the path containing `$ISALIST` points to an available implementation of the library. Using platform-specific or instruction set–specific versions of numerically intensive

applications can have a dramatic impact on the overall performance of an application on Oracle Solaris.

Open Source Software Libraries

Most of the popular open source and GNU software is either already available on Oracle Solaris 11 as IPS packages, or the ported binaries as well as the source code are available through various popular open source repositories. The techniques described in the previous section can be used to decide the version of libraries or commands to be used on a given installation to get the best performance on an Oracle Solaris 11 platform.

The Table 4-2 maps various popular GNU commands on RHEL with the equivalent ones on Oracle Solaris 11. The table also provides the IPS package name that needs to be installed to make the GNU utilities available on Oracle Solaris 11.

TABLE 4-2. GNU COMMANDS MAPPING AND AVAILABILITY

COMMAND	VERSION ON RHEL	LOCATION ON ORACLE SOLARIS 11	VERSION ON ORACLE SOLARIS 11	PACKAGE NAME ON ORACLE SOLARIS 11
`echo`	GNU coreutils 8.4	/usr/bin/gecho Links to /usr/gnu/bin/echo	GNU coreutils 8.5	file/gnu-coreutils
`FALSE`	GNU coreutils 8.4	/usr/bin/gfalse Links to /usr/gnu/bin/false	GNU coreutils 8.5	file/gnu-coreutils
`autopoint`	GNU gettext-tools 0.17	/usr/bin/autopoint	GNU gettext-tools 0.16.1	text/gnu-gettext
`gettextize`	GNU gettext-tools 0.17	/usr/bin/gettextize	GNU gettext-tools 0.16.1	text/gnu-gettext
`glib-gettextize`	GNU glib 2.22.5	/usr/bin/glib-gettextize	GNU glib 2.28.6	library/glib2
`autoconf`	GNU Autoconf2.63	/usr/bin/autoconf	GNU Autoconf2.63	developer/build/autoconf
`autoheader`	GNU Autoconf2.63	/usr/bin/autoheader	GNU Autoconf2.63	developer/build/autoconf
`autom4te`	GNU Autoconf2.63	/usr/bin/autom4te	GNU Autoconf2.63	developer/build/autoconf
`autoreconf`	GNU Autoconf2.63	/usr/bin/autoreconf	GNU Autoconf2.63	developer/build/autoconf
`autoscan`	GNU Autoconf2.63	/usr/bin/autoscan	GNU Autoconf2.63	developer/build/autoconf
`autoupdate`	GNU Autoconf2.63	/usr/bin/autoupdate	GNU Autoconf2.63	developer/build/autoconf
`b2m`	GNU Emacs 23.1	/usr/bin/b2m	GNU Emacs 23.1	editor/gnu-emacs
`base64`	GNU coreutils 8.4	/usr/bin/base64	GNU coreutils 8.5	runtime/ruby-18
`basename`	GNU coreutils 8.4	/usr/bin/gbasename Links to /usr/gnu/bin/basename	GNU coreutils 8.5	file/gnu-coreutils
`bison`	GNU Bison 2.4.1	/usr/bin/bison	GNU Bison 2.3	developer/parser/bison

TABLE 4-2. GNU COMMANDS MAPPING AND AVAILABILITY

COMMAND	VERSION ON RHEL	LOCATION ON ORACLE SOLARIS 11	VERSION ON ORACLE SOLARIS 11	PACKAGE NAME ON ORACLE SOLARIS 11
`cat`	GNU coreutils 8.4	/usr/bin/gcat Links to /usr/gnu/bin/cat	GNU coreutils 8.5	file/gnu-coreutils
`chgrp`	GNU coreutils 8.4	/usr/bin/gchgrp Links to /usr/gnu/bin/chgrp	GNU coreutils 8.5	file/gnu-coreutils
`chmod`	GNU coreutils 8.4	/usr/bin/gchmod Links to /usr/gnu/bin/chmod	GNU coreutils 8.5	file/gnu-coreutils
`chown`	GNU coreutils 8.4	/usr/bin/gchown Links to /usr/gnu/bin/chown	GNU coreutils 8.5	file/gnu-coreutils
`chroot`	GNU coreutils 8.4	/usr/bin/gchroot Links to /usr/gnu/bin/chroot	GNU coreutils 8.5	file/gnu-coreutils
`cmp`	GNU diffutils 2.8.1	/usr/bin/gcmp Links to /usr/gnu/bin/cmp	GNU diffutils 2.8.7	text/gnu-diffutils
`comm`	GNU coreutils 8.4	/usr/bin/gcomm Links to /usr/gnu/bin/comm	GNU coreutils 8.5	file/gnu-coreutils
`cp`	GNU coreutils 8.4	/usr/bin/gcp Links to /usr/gnu/bin/cp	GNU coreutils 8.5	file/gnu-coreutils
`csplit`	GNU coreutils 8.4	/usr/bin/gcsplit Links to /usr/gnu/bin/csplit	GNU coreutils 8.5	file/gnu-coreutils
`cut`	GNU coreutils 8.4	/usr/bin/gcut Links to /usr/gnu/bin/cut	GNU coreutils 8.5	file/gnu-coreutils
`date`	GNU coreutils 8.4	/usr/bin/gdate Links to /usr/gnu/bin/date	GNU coreutils 8.5	file/gnu-coreutils
`df`	GNU coreutils 8.4	/usr/bin/gdf Links to /usr/gnu/bin/df	GNU coreutils 8.5	file/gnu-coreutils
`diff`	GNU diffutils 2.8.1	/usr/bin/gdiff Links to /usr/gnu/bin/diff	GNU diffutils 2.8.7	text/gnu-diffutils
`diff3`	GNU diffutils 2.8.1	/usr/bin/gdiff3 Links to /usr/gnu/bin/diff3	GNU diffutils 2.8.7	text/gnu-diffutils
`dir`	GNU coreutils 8.4	/usr/bin/dir	GNU coreutils 8.5	runtime/ruby-18

TABLE 4-2. GNU COMMANDS MAPPING AND AVAILABILITY

COMMAND	VERSION ON RHEL	LOCATION ON ORACLE SOLARIS 11	VERSION ON ORACLE SOLARIS 11	PACKAGE NAME ON ORACLE SOLARIS 11
`dircolors`	GNU coreutils 8.4	/usr/bin/dircolors	GNU coreutils 8.5	file/gnu-coreutils
`dirname`	GNU coreutils 8.4	/usr/bin/gdirname Links to /usr/gnu/bin/dirname	GNU coreutils 8.5	file/gnu-coreutils
`du`	GNU coreutils 8.4	/usr/bin/gdu Links to /usr/gnu/bin/du	GNU coreutils 8.5	file/gnu-coreutils
`env`	GNU coreutils 8.4	/usr/bin/genv Links to /usr/gnu/bin/env	GNU coreutils 8.5	file/gnu-coreutils
`envsubst`	GNU gettext-runtime 0.17	/usr/bin/envsubst	GNU gettext-runtime 0.16.1	text/gnu-gettext
`expand`	GNU coreutils 8.4	/usr/bin/gexpand Links to /usr/gnu/bin/expand	GNU coreutils 8.5	file/gnu-coreutils
`expr`	GNU coreutils 8.4	/usr/bin/gexpr Links to /usr/gnu/bin/expr	GNU coreutils 8.5	file/gnu-coreutils
`factor`	GNU coreutils 8.4	/usr/bin/gfactor Links to /usr/gnu/bin/factor	GNU coreutils 8.5	file/gnu-coreutils
`find`	GNU findutils 4.4.2	/usr/bin/gfind Links to /usr/gnu/bin/find	GNU find version 4.2.31	file/gnu-findutils
`fmt`	GNU coreutils 8.4	/usr/bin/gfmt Links to /usr/gnu/bin/fmt	GNU coreutils 8.5	file/gnu-coreutils
`fold`	GNU coreutils 8.4	/usr/bin/gfold Links to /usr/gnu/bin/fold	GNU coreutils 8.5	file/gnu-coreutils
`gettext`	GNU gettext-runtime 0.17	/usr/bin/ggettext Links to /usr/gnu/bin/gettext	GNU gettext-runtime 0.16.1	text/gnu-gettext
`addr2line`	GNU addr2line version 2.20.51.0.2-5.19.el6 20091009	/usr/bin/gaddr2line Links to /usr/gnu/bin/addr2line	GNU Binutils 2.19	developer/gnu-binutils
`as`	GNU assembler version 2.20.51.0.2-5.19.el6 20091009	/usr/bin/gas Links to /usr/gnu/bin/as	GNU Binutils 2.19	developer/gnu-binutils
`awk`	GNU Awk 3.1.7	/usr/bin/gawk Links to /usr/gnu/bin/awk	GNU Awk 3.1.5	text/gawk

TABLE 4-2. GNU COMMANDS MAPPING AND AVAILABILITY

COMMAND	VERSION ON RHEL	LOCATION ON ORACLE SOLARIS 11	VERSION ON ORACLE SOLARIS 11	PACKAGE NAME ON ORACLE SOLARIS 11
`pgawk`	GNU Awk 3.1.7	/usr/bin/pgawk	GNU Awk 3.1.5	text/gawk
`bash`	GNU bash, version 4.1.2(1)-release (x86_64-redhat-linux-gnu)	/usr/bin/bash	GNU bash version 4.1.9(1)-release (i386-pc-solaris2.11)	shell/bash
`gc++filt`	GNU c++filt version 2.20.51.0.2-5.19.el6 20091009	/usr/bin/gc++filt Links to /usr/gnu/bin/c++filt	GNU Binutils 2.19	developer/gnu-binutils
`emacs`	GNU Emacs 23.1.1	/usr/bin/emacs	GNU Emacs 23.1.1	library/desktop/gtk2
`prof`	GNU gprof version 2.20.51.0.2-5.19.el6 20091009	/usr/bin/ggprof Links to /usr/gnu/bin/gprof	GNU Binutils 2.19	developer/gnu-binutils
`egrep`	GNU grep 2.6.3	/usr/bin/gegrep Links to /usr/gnu/bin/egrep	GNU grep 2.5.4	text/gnu-grep
`fgrep`	GNU grep 2.6.3	/usr/bin/gfgrep Links to /usr/gnu/bin/fgrep	GNU grep 2.5.4	text/gnu-grep
`grep`	GNU grep 2.6.4	/usr/bin/ggrep Links to /usr/gnu/bin/grep	GNU grep 2.5.4	text/gnu-grep
`gimp-2.6`	GNU Image Manipulation Program version 2.6.10	/usr/bin/gimp-2.6	GNU Image Manipulation Program version 2.6.10	image/editor/gimp
`gimp-console`	GNU Image Manipulation Program version 2.6.11	/usr/bin/gimp-console Links to gimp-console-2.6	GNU Image Manipulation Program version 2.6.10	image/editor/gimp
`gimp-console-2.6`	GNU Image Manipulation Program version 2.6.12	/usr/bin/gimp-console-2.6	GNU Image Manipulation Program version 2.6.10	image/editor/gimp
`gimp`	GNU Image Manipulation Program version 2.6.9	/usr/bin/gimp Links to gimp-2.6	GNU Image Manipulation Program version 2.6.10	library/desktop/gtk2
`gld`	GNU ld version 2.20.51.0.2-5.19.el6 20091009	/usr/bin/gld Links to /usr/gnu/bin/ld	GNU Binutils 2.19	system/kernel
`gmake`	GNU Make 3.81	/usr/bin/gmake Links to /usr/gnu/bin/make	GNU Make 3.81	developer/build/gnu-make

TABLE 4-2. GNU COMMANDS MAPPING AND AVAILABILITY

COMMAND	VERSION ON RHEL	LOCATION ON ORACLE SOLARIS 11	VERSION ON ORACLE SOLARIS 11	PACKAGE NAME ON ORACLE SOLARIS 11
`rnano`	GNU mano version 2.0.10	/usr/bin/rnano	GNU mano version 2.0.9	editor/nano
`nano`	GNU nano version 2.0.9	/usr/bin/nano	GNU nano version 2.0.9	editor/nano
`nm`	GNU nm version 2.20.51.0.2-5.19.el6 20091009	/usr/bin/gnm Links to /usr/gnu/bin/nm	GNU Binutils 2.19	developer/gnu-binutils
`objcopy`	GNU objcopy version 2.20.51.0.2-5.19.el6 20091009	/usr/bin/gobjcopy Links to /usr/gnu/bin/objcopy	GNU Binutils 2.19	developer/gnu-binutils
`objdump`	GNU objdump version 2.20.51.0.2-5.19.el6 20091009	/usr/bin/gobjdump Links to /usr/gnu/bin/objdump	GNU Binutils 2.19	developer/gnu-binutils
`readelf`	GNU readelf version 2.20.51.0.2-5.19.el6 20091009	/usr/bin/greadelf Links to /usr/gnu/bin/readelf	GNU Binutils 2.19	developer/gnu-binutils
`sed`	GNU sed version 4.2.1	/usr/bin/gsed Links to /usr/gnu/bin/sed	GNU sed version 4.2.1	text/gnu-sed
`size`	GNU size version 2.20.51.0.2-5.19.el6 20091009	/usr/bin/gsize Links to /usr/gnu/bin/size	GNU Binutils 2.19	developer/gnu-binutils
`strings`	GNU strings version 2.20.51.0.2-5.19.el6 20091009	/usr/bin/gstrings Links to /usr/gnu/bin/strings	GNU Binutils 2.19	developer/gnu-binutils
`strip`	GNU strip version 2.20.51.0.2-5.19.el6 20091009	/usr/bin/gstrip Links to /usr/gnu/bin/strip	GNU Binutils 2.19	developer/gnu-binutils
`wget`	GNU Wget 1.12 built on Linux-gnu.	/usr/bin/wget	GNU Wget 1.12 built on solaris2.11.	web/wget
`which`	GNU which v2.19	/usr/bin/gwhich Links to /usr/gnu/bin/which	GNU which v2.16	shell/which
`groups`	GNU coreutils 8.4	/usr/bin/ggroups Links to /usr/gnu/bin/groups	GNU coreutils 8.5	file/gnu-coreutils
`head`	GNU coreutils 8.4	/usr/bin/ghead Links to /usr/gnu/bin/head	GNU coreutils 8.5	file/gnu-coreutils

TABLE 4-2. GNU COMMANDS MAPPING AND AVAILABILITY

COMMAND	VERSION ON RHEL	LOCATION ON ORACLE SOLARIS 11	VERSION ON ORACLE SOLARIS 11	PACKAGE NAME ON ORACLE SOLARIS 11
`hostid`	GNU coreutils 8.4	/usr/bin/ghostid Links to /usr/gnu/bin/hostid	GNU coreutils 8.5	file/gnu-coreutils
`id`	GNU coreutils 8.4	/usr/bin/gid Links to /usr/gnu/bin/id	GNU coreutils 8.5	runtime/ruby-18
`idn`	GNU Libidn 1.18	/usr/bin/idn	(GNU Libidn) 1.19	web/php-52/extension/php-idn
`ifnames`	GNU Autoconf2.63	/usr/bin/ifnames	GNU Autoconf2.63	developer/build/autoconf
`info`	GNU texinfo 4.13	/usr/bin/info	GNU texinfo 4.7	runtime/ruby-18
`infokey`	GNU texinfo 4.13	/usr/bin/infokey	GNU texinfo 4.7	text/texinfo
`install`	GNU coreutils 8.4	/usr/bin/ginstall Links to /usr/gnu/bin/install	GNU coreutils 8.5	file/gnu-coreutils
`install-info`	GNU texinfo 4.13	/usr/bin/install-info	GNU texinfo 4.7	text/texinfo
`intltoolize`	GNU intltool 0.41.0	/usr/bin/intltoolize	GNU intltool 0.40.6	developer/gnome/gettext
`join`	GNU coreutils 8.4	/usr/bin/gjoin Links to /usr/gnu/bin/join	GNU coreutils 8.5	file/gnu-coreutils
`link`	GNU coreutils 8.4	/usr/bin/glink Links to /usr/gnu/bin/link	GNU coreutils 8.5	file/gnu-coreutils
`ln`	GNU coreutils 8.4	/usr/bin/gln Links to /usr/gnu/bin/ln	GNU coreutils 8.5	file/gnu-coreutils
`logname`	GNU coreutils 8.4	/usr/bin/glogname Links to /usr/gnu/bin/logname	GNU coreutils 8.5	file/gnu-coreutils
`ls`	GNU coreutils 8.4	/usr/bin/gls Links to /usr/gnu/bin/ls	GNU coreutils 8.5	file/gnu-coreutils
`m4`	GNU M4 1.4.13	/usr/bin/gm4 Links to /usr/gnu/bin/m4	GNU M4 1.4.12	developer/macro/gnu-m4
`makeinfo`	GNU texinfo 4.13	/usr/bin/makeinfo	GNU texinfo 4.7	text/texinfo
`md5sum`	GNU coreutils 8.4	/usr/bin/md5sum	GNU coreutils 8.5	file/gnu-coreutils
`mkdir`	GNU coreutils 8.4	/usr/bin/gmkdir Links to /usr/gnu/bin/mkdir	GNU coreutils 8.5	file/gnu-coreutils

TABLE 4-2. GNU COMMANDS MAPPING AND AVAILABILITY

COMMAND	VERSION ON RHEL	LOCATION ON ORACLE SOLARIS 11	VERSION ON ORACLE SOLARIS 11	PACKAGE NAME ON ORACLE SOLARIS 11
`mkfifo`	GNU coreutils 8.4	/usr/bin/gmkfifo Links to /usr/gnu/bin/mkfifo	GNU coreutils 8.5	file/gnu-coreutils
`mknod`	GNU coreutils 8.4	/usr/bin/gmknod Links to /usr/gnu/bin/mknod	GNU coreutils 8.5	file/gnu-coreutils
`mktemp`	GNU coreutils 8.4	/usr/bin/gmktemp Links to /usr/gnu/bin/mktemp	GNU coreutils 8.5	file/gnu-coreutils
`msgattrib`	GNU gettext-tools 0.17	/usr/bin/msgattrib	GNU gettext-tools 0.16.1	text/gnu-gettext
`msgcat`	GNU gettext-tools 0.17	/usr/bin/msgcat	GNU gettext-tools 0.16.1	text/gnu-gettext
`msgcmp`	GNU gettext-tools 0.17	/usr/bin/msgcmp	GNU gettext-tools 0.16.1	text/gnu-gettext
`msgcomm`	GNU gettext-tools 0.17	/usr/bin/msgcomm	GNU gettext-tools 0.16.1	text/gnu-gettext
`msgconv`	GNU gettext-tools 0.17	/usr/bin/msgconv	GNU gettext-tools 0.16.1	text/gnu-gettext
`msgen`	GNU gettext-tools 0.17	/usr/bin/msgen	GNU gettext-tools 0.16.1	text/gnu-gettext
`msgexec`	GNU gettext-tools 0.17	/usr/bin/msgexec	GNU gettext-tools 0.16.1	text/gnu-gettext
`msgfilter`	GNU gettext-tools 0.17	/usr/bin/msgfilter	GNU gettext-tools 0.16.1	text/gnu-gettext
`msgfmt`	GNU gettext-tools 0.17	/usr/bin/gmsgfmt Links to /usr/gnu/bin/msgfmt	GNU gettext-tools 0.16.1	text/locale
`msggrep`	GNU gettext-tools 0.17	/usr/bin/msggrep	GNU gettext-tools 0.16.1	text/gnu-gettext
`msginit`	GNU gettext-tools 0.17	/usr/bin/msginit	GNU gettext-tools 0.16.1	text/gnu-gettext
`msgmerge`	GNU gettext-tools 0.17	/usr/bin/msgmerge	GNU gettext-tools 0.16.1	developer/gnome/gettext
`msgunfmt`	GNU gettext-tools 0.17	/usr/bin/msgunfmt	GNU gettext-tools 0.16.1	text/gnu-gettext
`msguniq`	GNU gettext-tools 0.17	/usr/bin/msguniq	GNU gettext-tools 0.16.1	text/gnu-gettext
`mv`	GNU coreutils 8.4	/usr/bin/gmv Links to /usr/gnu/bin/mv	GNU coreutils 8.5	file/gnu-coreutils
`ngettext`	GNU gettext-runtime 0.17	/usr/bin/ngettext	GNU gettext-runtime 0.16.1	text/gnu-gettext
`nice`	GNU coreutils 8.4	/usr/bin/gnice Links to /usr/gnu/bin/nice	GNU coreutils 8.5	file/gnu-coreutils
`nl`	GNU coreutils 8.4	/usr/bin/gnl Links to /usr/gnu/bin/nl	GNU coreutils 8.5	file/gnu-coreutils

TABLE 4-2. GNU COMMANDS MAPPING AND AVAILABILITY

COMMAND	VERSION ON RHEL	LOCATION ON ORACLE SOLARIS 11	VERSION ON ORACLE SOLARIS 11	PACKAGE NAME ON ORACLE SOLARIS 11
`nohup`	GNU coreutils 8.4	/usr/bin/gnohup Links to /usr/gnu/bin/nohup	GNU coreutils 8.5	file/gnu-coreutils
`autogen`	-	/usr/bin/autogen	GNU AutoGen - The Automated Program Generator - Ver. 5.9	developer/build/autogen
`card`	-	/usr/bin/card	GNU a2ps 4.14	driver/pcmcia
`columns`	-	/usr/bin/columns	GNU AutoGen - Columnize Input Text - Ver. 1.1	terminal/resize
`fixps`	-	/usr/bin/fixps	GNU a2ps 4.14	print/filter/a2ps
`a2ps`	-	/usr/bin/a2ps	GNU a2ps 4.14	print/filter/a2ps
`nproc`	GNU coreutils 8.4	/usr/bin/nproc	GNU coreutils 8.5	file/gnu-coreutils
`od`	GNU coreutils 8.4	/usr/bin/god Links to /usr/gnu/bin/od	GNU coreutils 8.5	file/gnu-coreutils
`paste`	GNU coreutils 8.4	/usr/bin/gpaste Links to /usr/gnu/bin/paste	GNU coreutils 8.5	file/gnu-coreutils
`pathchk`	GNU coreutils 8.4	/usr/bin/gpathchk Links to /usr/gnu/bin/pathchk	GNU coreutils 8.5	file/gnu-coreutils
`perf`	perf version 2.6.32-122.el6.x86_64	/usr/bin/gperf	GNU gperf 3.0.3	developer/gperf
`pinky`	GNU coreutils 8.4	/usr/bin/pinky	GNU coreutils 8.5	file/gnu-coreutils
`pr`	GNU coreutils 8.4	/usr/bin/gpr Links to /usr/gnu/bin/pr	GNU coreutils 8.5	file/gnu-coreutils
`printenv`	GNU coreutils 8.4	/usr/bin/printenv	GNU coreutils 8.5	web/server/apache-22
`ptx`	GNU coreutils 8.4	/usr/bin/ptx	GNU coreutils 8.5	file/gnu-coreutils
`readlink`	GNU coreutils 8.4	/usr/bin/readlink	GNU coreutils 8.5	file/gnu-coreutils
`rm`	GNU coreutils 8.4	/usr/bin/grm Links to /usr/gnu/bin/rm	GNU coreutils 8.5	file/gnu-coreutils
`rmdir`	GNU coreutils 8.4	/usr/bin/grmdir Links to /usr/gnu/bin/rmdir	GNU coreutils 8.5	file/gnu-coreutils

TABLE 4-2. GNU COMMANDS MAPPING AND AVAILABILITY

COMMAND	VERSION ON RHEL	LOCATION ON ORACLE SOLARIS 11	VERSION ON ORACLE SOLARIS 11	PACKAGE NAME ON ORACLE SOLARIS 11
`sdiff`	GNU diffutils 2.8.1	/usr/bin/gsdiff Links to /usr/gnu/bin/sdiff	GNU diffutils 2.8.7	text/gnu-diffutils
`seq`	GNU coreutils 8.4	/usr/bin/seq	GNU coreutils 8.5	runtime/ruby-18
`sha1sum`	GNU coreutils 8.4	/usr/bin/sha1sum	GNU coreutils 8.5	file/gnu-coreutils
`sha224sum`	GNU coreutils 8.4	/usr/bin/sha224sum	GNU coreutils 8.5	file/gnu-coreutils
`sha256sum`	GNU coreutils 8.4	/usr/bin/sha256sum	GNU coreutils 8.5	file/gnu-coreutils
`sha384sum`	GNU coreutils 8.4	/usr/bin/sha384sum	GNU coreutils 8.5	file/gnu-coreutils
`sha512sum`	GNU coreutils 8.4	/usr/bin/sha512sum	GNU coreutils 8.5	file/gnu-coreutils
`shred`	GNU coreutils 8.4	/usr/bin/shred	GNU coreutils 8.5	file/gnu-coreutils
`shuf`	GNU coreutils 8.4	/usr/bin/shuf	GNU coreutils 8.5	file/gnu-coreutils
`sleep`	GNU coreutils 8.4	/usr/bin/gsleep Links to /usr/gnu/bin/sleep	GNU coreutils 8.5	file/gnu-coreutils
`sort`	GNU coreutils 8.4	/usr/bin/gsort Links to /usr/gnu/bin/sort	GNU coreutils 8.5	file/gnu-coreutils
`split`	GNU coreutils 8.4	/usr/bin/gsplit Links to /usr/gnu/bin/split	GNU coreutils 8.5	file/gnu-coreutils
`stat`	GNU coreutils 8.4	/usr/bin/stat	GNU coreutils 8.5	runtime/ruby-18
`stdbuf`	GNU coreutils 8.4	/usr/bin/stdbuf	GNU coreutils 8.5	file/gnu-coreutils
`stty`	GNU coreutils 8.4	/usr/bin/gstty Links to /usr/gnu/bin/stty	GNU coreutils 8.5	file/gnu-coreutils
`sum`	GNU coreutils 8.4	/usr/bin/gsum Links to /usr/gnu/bin/sum	GNU coreutils 8.5	file/gnu-coreutils
`tac`	GNU coreutils 8.4	/usr/bin/tac	GNU coreutils 8.5	file/gnu-coreutils
`tail`	GNU coreutils 8.4	/usr/bin/gtail Links to /usr/gnu/bin/tail	GNU coreutils 8.5	file/gnu-coreutils
`tee`	GNU coreutils 8.4	/usr/bin/gtee Links to /usr/gnu/bin/tee	GNU coreutils 8.5	file/gnu-coreutils
`timeout`	GNU coreutils 8.4	/usr/bin/timeout	GNU coreutils 8.5	runtime/ruby-18
`touch`	GNU coreutils 8.4	/usr/bin/gtouch Links to /usr/gnu/bin/touch	GNU coreutils 8.5	file/gnu-coreutils

TABLE 4-2. GNU COMMANDS MAPPING AND AVAILABILITY

COMMAND	VERSION ON RHEL	LOCATION ON ORACLE SOLARIS 11	VERSION ON ORACLE SOLARIS 11	PACKAGE NAME ON ORACLE SOLARIS 11
`tr`	GNU coreutils 8.4	/usr/bin/gtr Links to /usr/gnu/bin/tr	GNU coreutils 8.5	file/gnu-coreutils
`truncate`	GNU coreutils 8.4	/usr/bin/truncate	GNU coreutils 8.5	file/gnu-coreutils
`tty`	GNU coreutils 8.4	/usr/bin/gtty Links to /usr/gnu/bin/tty	GNU coreutils 8.5	file/gnu-coreutils
`uname`	GNU coreutils 8.4	/usr/bin/guname Links to /usr/gnu/bin/uname	GNU coreutils 8.5	file/gnu-coreutils
`unexpand`	GNU coreutils 8.4	/usr/bin/gunexpand Links to /usr/gnu/bin/unexpand	GNU coreutils 8.5	file/gnu-coreutils
`uniq`	GNU coreutils 8.4	/usr/bin/guniq Links to /usr/gnu/bin/uniq	GNU coreutils 8.5	file/gnu-coreutils
`unlink`	GNU coreutils 8.4	/usr/bin/gunlink Links to /usr/gnu/bin/unlink	GNU coreutils 8.5	file/gnu-coreutils
`users`	GNU coreutils 8.4	/usr/bin/users	GNU coreutils 8.5	network/chat/ircii
`vdir`	GNU coreutils 8.4	/usr/bin/vdir	GNU coreutils 8.5	file/gnu-coreutils
`wc`	GNU coreutils 8.4	/usr/bin/gwc Links to /usr/gnu/bin/wc	GNU coreutils 8.5	file/gnu-coreutils
`who`	GNU coreutils 8.4	/usr/bin/gwho Links to /usr/gnu/bin/who	GNU coreutils 8.5	file/gnu-coreutils
`whoami`	GNU coreutils 8.4	/usr/bin/gwhoami Links to /usr/gnu/bin/whoami	GNU coreutils 8.5	file/gnu-coreutils
`whoami`	GNU coreutils 8.4	/usr/bin/whoami	GNU coreutils 8.5	file/gnu-coreutils
`xargs`	GNU findutils 4.4.2	/usr/bin/gxargs Links to /usr/gnu/bin/xargs	GNU xargs version 4.2.31	file/gnu-findutils
`xgettext`	GNU gettext-tools 0.17	/usr/bin/gxgettext Links to /usr/gnu/bin/xgettext	GNU gettext-tools 0.16.1	text/gnu-gettext
`yes`	GNU coreutils 8.4	/usr/bin/gyes Links to /usr/gnu/bin/yes	GNU coreutils 8.5	file/gnu-coreutils

TABLE 4-2. GNU COMMANDS MAPPING AND AVAILABILITY

COMMAND	VERSION ON RHEL	LOCATION ON ORACLE SOLARIS 11	VERSION ON ORACLE SOLARIS 11	PACKAGE NAME ON ORACLE SOLARIS 11
`clisp`	-	/usr/bin/clisp	GNU CLISP 2.47 (2008-10-23) (built on userland [10.134.73.164])	runtime/clisp
`diffmk`	-	/usr/bin/gdiffmk Links to /usr/gnu/bin/diffmk	GNU gdiffmk (groff) version 1.19.2	text/groff/groff-core
`grolbp`	GNU grolbp /groff version 1.18,1,4	/usr/bin/grolbp	GNU grolbp (groff) version 1.19.2	text/groff
`pth-config`	-	/usr/bin/pth-config	GNU Pth 2.0.7	library/pth
`ranlib`	-	/usr/bin/granlib Links to /usr/gnu/bin/ranlib	GNU ranlib GNU Binutils 2.19	developer/gnu-binutils
`kill`	-	/usr/bin/gkill Links to /usr/gnu/bin/kill	GNU coreutils 8.5	file/gnu-coreutils
`pdiff`	-	/usr/bin/pdiff	GNU a2ps 4.14	print/filter/a2ps
`printf`	-	/usr/bin/gprintf Links to /usr/gnu/bin/printf	GNU coreutils 8.5	file/gnu-coreutils
`psmandup`	-	/usr/bin/psmandup	GNU a2ps 4.14	print/filter/a2ps
`psset`	-	/usr/bin/psset	GNU a2ps 4.14	print/filter/a2ps
`pwd`	-	/usr/bin/gpwd Links to /usr/gnu/bin/pwd	GNU coreutils 8.5	file/gnu-coreutils
`texi2dvi4a2ps`	-	/usr/bin/texi2dvi4a2ps	GNU Texinfo 4.1 - 1.1.1.1.2.3	print/filter/a2ps
`texi2dvi`	-	/usr/bin/texi2dvi	GNU Texinfo 4.7 1.3	text/texinfo
`texindex`	-	/usr/bin/texindex	GNU texinfo 4.7	text/texinfo
`true`	GNU coreutils 8.4	/usr/bin/gtrue Links to /usr/gnu/bin/true	GNU coreutils 8.5	file/gnu-coreutils
`uptime`	procps version 3.2.8	/usr/bin/guptime Links to /usr/gnu/bin/uptime	GNU coreutils 8.5	file/gnu-coreutils
`xml2ag`	-	/usr/bin/xml2ag	GNU AutoGen - XML to AutoGen Definiton Converter - Ver. 5.9	developer/build/autogen

Debugging Applications

Many powerful tools are available on Oracle Solaris 11 for debugging software. These debugging tools can be broadly categorized as follows:

- Kernel-land debuggers, for running in kernel mode
  - Kernel mode maps to the privilege mode of the CPU. Examples of kernel modules include file system kernel modules, device drivers, and so on.
- User-land debuggers for running in user mode
  - User-land maps to the non-privilege mode of the CPU. User-land is where applications execute and the applications interface to the OS kernel via system calls. Kernel-land code and user-land code have different address spaces.
- Tools for addressing specific issues
  - This includes tools for detecting memory-access errors or leaks, code coverage, data races and deadlocks, hot locks, and so on.

Kernel Debuggers

The tools under this category are used for debugging code running in kernel mode (which maps to the privilege mode of the CPU). The most popular tools among kernel developers are `kdb` and `mdb`:

- `kdb` is the interactive kernel debugger.
- `mdb` is the modular debugger generally used for debugging the live kernel, processes, core dumps, memory leaks, and so on, but it can also be used for debugging application core dumps. `mdb` is a very popular tool for debugging system crash dumps. It is an easy-to-extend utility for low-level debugging and it is modularized. Oracle Solaris 11 includes a set of `mdb` modules that assist programmers in debugging the Oracle Solaris kernel and related device drivers. Third-party developers can develop their own debugging modules for supervisor or user software.

Source-Level Debugging with `dbx`

Similar to `gdb` (the GNU debugger on Linux), `dbx` is a proprietary source-level debugger shipped with Oracle Solaris Studio. `dbx` can be used for source-level debugging and execution of programs written in C++, ANSI C, Fortran 77, Fortran 95, and Java programming languages. It provides symbolic debugging for programs written in C and C++. Useful features of `dbx` include stepping through programs one source line or one machine instruction at a time. In addition to simply viewing the operation of the program, variables can be manipulated and a wide range of expressions can be evaluated and displayed. A program, a running process, or a core file can be debugged using `dbx`. When `dbx` is used to debug a running process, the process is stopped.

To use `dbx` effectively, you need a debug binary, that is, you need to build your source code with the `-g` compiler flag. Also note that the full debugging information is available only with *no compiler optimization*. Only partial debugging support is available for optimized code. `dbx` allows you to modify

and recompile a native source file and continue executing without rebuilding the entire program. You can resume running from the *code fix* location. You do not need to relink or reload to continue debugging. In `dbx`, you can also navigate between threads, suspend, step a thread, and display the stack and locks.

Tools for Addressing Specific Issues

- Memory access errors or leaks (`discover`, `mdb libumem`): There are number of `dbx` commands for detecting memory leaks. For example, the `dbx check memusage` command enables memory leak detection and memory usage summary reports at program exit. The `showleaks` command shows memory leaks when you've stopped at a breakpoint, and you can use the `dbx showmemuse` command to see the memory usage report when you've stopped at a breakpoint. Through `dbxenv` variables `rtc_error_log_file_name` and `rtc_mel_at_exit`, you can specify where you want the summary report written to and control the verbosity of the memory usage report at the end of execution.
- Hot locks (`lockstat`, `plockstat`): The `lockstat` and `plockstat` tools are built-in Oracle Solaris 11 tools. `plockstat` is used to find locks in applications, while `lockstat` will report kernel lock and profiling statistics.
- Code coverage (`uncover`): `uncover` is a command-line tool for measuring the code coverage of user applications. This tool can display information about areas of the application being exercised during testing. The coverage information reported by this tool could be at a function level, statement level, block level, or instruction level.
- Data races and deadlocks (Thread Analyzer): You can use Thread Analyzer to nail down difficult-to-detect code issues in multithreaded programs. For example, it helps you detect data races and deadlock conditions.

Identifying Issues Using DTrace

DTrace is a comprehensive dynamic tracing framework for troubleshooting kernel and application problems on production systems in real time. DTrace can be used to get a global overview of a running system, such as the amount of memory, the CPU time, and file system and network resources used by the active processes. It can also provide much more fine-grained information, such as a log of the arguments with which a specific function is being called or a list of the processes accessing a specific file. It is a very powerful dynamic tracing tool that can trace any part of the system. You can instrument the system by writing a simple Dtrace script. The tracing gets enabled only for the code area (probe) mentioned in the Dtrace script. Hence, there is near zero overhead on the system when the probes are not enabled. This infrastructure can be safely used in production (assuming the user knows the possible overhead and impact of the script.)

Please refer to the following resources for additional information about DTrace tools:

- DTrace toolkit, which is a set of ready-to-use DTrace scripts written by Brendan Gregg and available for free download at http://hub.opensolaris.org/bin/view/Community+Group+dtrace/dtracetoolkit.

- `plockstat`, which is a tool implemented using DTrace to print user-level lock statistics. This tool is very useful for analyzing and zeroing on lock contention issues in an application.

Chapter 5 Threads and Multiprocessing

POSIX Compliance

The standard POSIX.1c threads extensions (IEEE Std 1003.1c-1995) defines APIs for creating and manipulating threads. *POSIX threads*, usually referred to as *Pthreads*, is a POSIX standard for threads. POSIX.1 specifies a set of interfaces (functions and header files) for threaded programming. Oracle Solaris Studio as well as GNU compilers support the POSIX threads programming model.

Linux Threading Model

On RHEL, two threading implementations have been provided over time by the GNU C library (`glibc`). Both of these are so-called 1:1 implementations, meaning that each thread maps to a kernel scheduling entity:

- LinuxThreads. This is the original Pthreads implementation. (Since `glibc` 2.4, this implementation is no longer supported.)
- NPTL (Native POSIX Threads Library). This is the modern Pthreads implementation.

By comparison with LinuxThreads, NPTL provides "closer" conformance to the requirements of the POSIX.1 specification and better performance when large numbers of threads are created. NPTL has been available since `glibc` 2.3.2 and requires features that are present in the Linux 2.6 kernel.

Since `glibc` 2.3.2, the `getconf` command can be used to determine a RHEL system's threading implementation, for example:

```
bash$ getconf GNU_LIBPTHREAD_VERSION
NPTL 2.12
```

With older `glibc` versions, a command such as the following should be sufficient to determine the default threading implementation:

```
bash$ $( ldd /bin/ls | grep libc.so | awk '{print $3}' ) | egrep -i 'threads|nptl'
```

where `nptl` refers to the native POSIX threads library by Ulrich Drepper et al.

Selecting the Thread Implementation

On systems with a `glibc` version that supports both LinuxThreads and NPTL (for example, `glibc` 2.3.x), the `LD_ASSUME_KERNEL` environment variable can be used to override the dynamic linker's default choice of threading implementation. This variable tells the dynamic linker to assume that it is running on top of a particular kernel version. By specifying a kernel version that does not provide the support required by NPTL, you can force the use of LinuxThreads. The most likely reason for doing this is to run a (broken) application that depends on some nonconformant behavior in LinuxThreads, for example:

```
bash$ $( LD_ASSUME_KERNEL=2.12 ldd /bin/ls | grep libc.so |awk '{print $3}' ) | egrep
      -i 'threads|ntpl'
```

Oracle Solaris Threading Model

Before the POSIX standard was ratified, the Oracle Solaris multithreading API was implemented in the Oracle Solaris `libthread` library, which was developed by Sun and later became the basis for the UNIX International (UI) threads standard.

- Oracle Solaris threads is a Sun Microsystems threads interface that is not POSIX threads–compliant. It's a predecessor of Pthreads. This implementation uses an MxN model (User X kernel threads). Based on performance studies and extensive workload analyses done for large enterprise customer workloads with very large numbers of processors, the MxN model was replaced with a 1:1 model starting with Oracle Solaris 9.
- Pthreads is a threads interface that is POSIX threads–compliant. Support for the POSIX standard was added with the `libpthread` API in the latter versions Oracle Solaris. The `libthread` and `libpthread` libraries were merged into the standard libc C library beginning in Oracle Solaris 10. Oracle Solaris uses a 1:1 (user:kernel) thread model in preference to the historic MxN implementation. By simplifying the underlying thread implementation, existing applications now can see dramatic performance and stability improvements without requiring recompilation.

While Oracle Solaris supports both Pthreads and Oracle Solaris threads, the Pthread model is recommended for new application development and porting efforts. Over 100 standards POSIX functions are available through the Pthreads API. See the `pthreads` man page for detailed information, including a comparison to the Oracle Solaris threading APIs.

It is possible to mix the two threading models, but this creates undue complexity and should be avoided. Both the `libthread` and `libpthread` libraries are maintained in Oracle Solaris to provide backward compatibility for both runtime and compilation environments. The `libthread.so.1` and `libpthread.so.1` shared objects are implemented as filters on `libc.so.1`. See the `libthread`(3LIB) and `libpthread`(3LIB) man pages for more information.

Differences Between the Oracle Solaris and Linux Threading Models

Overall, from the migration perspective, the implementation differences on the two platforms primarily fall into two categories:

- Nonstandard interfaces. In the nonstandard interfaces category, falls the entire implementation coming from LinuxThreads (on RHEL) and `libthread.so` (the Solaris non-POSIX thread library). The interfaces exposed from these thread libraries can to some extent be mapped to functions in the POSIX thread library (which Oracle Solaris also provides).
- Standard interface. Since RHEL 6.0 and Oracle Solaris implement the same POSIX 1003.1c standard, migration to Oracle Solaris would not involve too much rework. Note that you will have to

take into account some minor differences in the interface, which are largely due to edge cases left unspecified by the standard or by permitted implementation dependences.

Traditionally, both the platforms supported both POSIX and non-POSIX threads, but for new application development and porting efforts on the latest versions of both operating systems, the POSIX (Pthreads) model is recommended. Hence, this discussion is hereafter limited to POSIX-compliant implementations only.

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` has a priority attribute that RHEL `pthreads` lacks.) While most applications can do very well with the default scheduling, a large number of threads and mission-critical enterprise-class multiprocess applications can benefit from careful use of the various process scheduler features. For an introduction to the process scheduler, see http://download.oracle.com/docs/cd/E19963-01/html/821-1460/rmfss-2.html.As far as the schedular is concerned, it is worth noting that RHEL NPTL has one area of noncompliance: NPTL threads do not share a common nice value. Per POSIX standard, since all the threads are part of same process, they should have a common nice value.

TABLE 5-1. IMPLEMENTATION-LEVEL DIFFERENCES

FUNCTION	RHEL	ORACLE SOLARIS 11
`int pthread_attr_init(pthread_attr_t *attr);`	On Linux, these functions are always assumed to succeed.	The `pthread_attr_init()` function will fail with `ENOMEM`, if insufficient memory exists to initialize the thread attributes object.
`int pthread_attr_destroy(pthread_attr_t *attr);`		The `pthread_attr_destroy()` function might fail with `EINVAL` if `attr` is invalid.
`int pthread_atfork(void (*prepare)(void), void (*parent)(void), void (*child)(void));`	`#include <pthread.h>`	`#include <sys/types.h>` `#include <unistd.h>`
	The parent and child fork handlers shall be called in the order in which they were established by calls to `pthread_atfork()`. The prepare fork handlers shall be called in the opposite order.	The prepare fork handler is called in LIFO (last-in first-out) order, whereas the parent and child fork handlers are called in FIFO (first-in first-out) order. This calling order allows applications to preserve locking order.

TABLE 5-1. IMPLEMENTATION-LEVEL DIFFERENCES

FUNCTION	RHEL	ORACLE SOLARIS 11
`int pthread_cancel(pthread_t thread);`	On Linux, cancellation is implemented using signals. Under the NPTL threading implementation, the first real-time signal (that is, signal 32) is used for this purpose. On LinuxThreads, the second real-time signal is used if real-time signals are available; otherwise, `SIGUSR2` is used.	A thread acting on a cancellation request runs with all signals blocked. All thread termination functions, including cancellation cleanup handlers and thread-specific data destructor functions, are called with all signals blocked. The cancellation processing in `target_thread` runs asynchronously with respect to the calling thread returning from `pthread_cancel()`.
`void pthread_cleanup_push(void (*routine)(void *), void *arg);` `void pthread_cleanup_pop(int execute)`	On Linux, cancellation is implemented using signals. Under the NPTL threading implementation, the first real-time signal (tha this, signal 32) is used for this purpose. On LinuxThreads, the second real-time signal is used if real-time signals are available; otherwise, `SIGUSR2` is used.	An exiting or cancelled thread runs with all signals blocked. All thread termination functions, including cancellation cleanup handlers, are called with all signals blocked.
`pthread_create`	`int pthread_create(pthread_t *thread, const pthread_attr_t *attr, void *(*start_routine) (void *), void *arg);`	`int pthread_create(pthread_t *restrict thread, const pthread_attr_t *restrict attr, void *(*start_routine)(void*), void *restrict arg);`
`int pthread_kill(pthread_t thread, int sig);`	`#include <signal.h>`	`#include <signal.h>` `#include <pthread.h>`

TABLE 5-2. DEFAULT THREAD ATTRIBUTES

ATTRIBUTE	RHEL	ORACLE SOLARIS 11	COMMENT
Contention scope	`PTHREAD_SCOPE_SYSTEM`	`PTHREAD_SCOPE_PROCESS`	Resource competition within process
Detach state	`PTHREAD_CREATE_JOINABLE`	`PTHREAD_CREATE_JOINABLE`	Joinable by other threads
Scheduling priority	0	0	
Guard size	4096 bytes	`PAGESIZE`	Size of guard area for thread's stack
Stack size	0x201000 bytes	1 or 2 megabytes	
Policy	`SCHED_OTHER`	`SCHED_OTHER`	Traditional time-sharing policy thread
Inherit scheduler	`PTHREAD_INHERIT_SCHED`	`PTHREAD_INHERIT_SCHED`	Scheduling policy and parameters are inherited from the creating thread

Signals in Threaded Applications

The RHEL POSIX thread implementation is process-centric. On RHEL, as far as signals are concerned, each thread is viewed as an independent process (LWP). The RHEL kernel does not really differentiate between threads and processes. As a matter of fact, within a single process, the threads share many resources such as virtual memory, file descriptors, global variables, and so on. Because of this basic difference, the signals might not get handled correctly (in a POSIX–compliant way) in RHEL by multithreaded applications. In RHEL, the signals are directly sent to individual threads and not to the process as a whole (this is a deviation from the POSIX standard). The POSIX standard mandates that the signal received by a process can be handled by any single thread within the targeted process. Programs that depend on this aspect of signal delivery might find issues with this difference in implementation. It is important to note that this issue has already been addressed in the latest RHEL kernel patches, so whether a multithreaded application would face this issue is entirely dependent on the application's base development platform.

Getting the Most out of Oracle Solaris 11

Support for Latest Hardware Technologies

Historically, throughput has been a measure of the comparative effectiveness of large commercial computers that run many programs concurrently. When Oracle's family of SPARC processors based on chip multithreading (CMT) technology was first introduced, the concept of throughput computing was presented as a means to cut the cost and complexity of network computing. The idea is to use CMT-enabled processors to maximize application workload throughput. This means maximizing the aggregate amount of work done in a given amount of time on both a per-processor and a per-system basis.

By combining multicore processors with CMT technology, we can increase the overall throughput of the processor (socket) by the number of cores on the processor. This approach enables increased performance at a lower cost because fewer systems are required, which also means reduced power consumption, lower maintenance and administration costs, and increased reliability due to fewer systems.

In a CMT processor, a single core typically contains eight or more hardware threads. The CMT processors execute multiple software threads simultaneously on these hardware threads and this, in turn, can help improve price/performance and reduce the total cost of ownership. Unlike traditional single-threaded processors and even most current multicore processors, hardware multithreaded processors, such as Oracle's SPARC T4 processors, allow rapid switching between active threads as other threads stall for memory. On a CMT processor, 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, which are common in commercial workloads, benefit greatly from this model whether they comprise large multithreaded applications or large numbers of smaller single-threaded

applications. The number of simultaneous threads that can be accommodated is quite large. Oracle Solaris 11 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.

- The operating system is designed to take advantage of latest hardware trends.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.
- There is better system control. 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 (hardware threads). 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 the group.
- Oracle Solaris allows considerable flexibility for controlling process affinity. Processes and individual threads can be bound to either a processor or a processor set, if required or desired.

OpenMP Support

Support for OpenMP in Oracle Solaris Studio compilers means that the compilers can look for OpenMP directives (pragma) in the source code in order to build a parallel version of the application. Similar to automatic parallelization, an Oracle Solaris Studio compiler does the additional 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 version and a parallel version of the application from the exact same code base, which can be helpful for debugging. Several compiler flags are available in Oracle Solaris Studio to support OpenMP.

- You can enable OpenMP with the `-xopenmp` compiler flag.
- Directives are recognized only when the flag is used.
- You can set the `-xvpara` compiler flag to report potential parallelization issues.
- You can 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 variables 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.

Auto Parallelization and Compile-Time Optimizations

Traditionally, most code was developed without support for parallel threads of execution, making it difficult to fully exploit advancements in the lasted hardware technologies. The latest Oracle Solaris Studio compilers provide mechanisms to let applications run multiple threads without requiring the

developer to specify how. Within legacy code, execution loops, in particular, often represent opportunities for optimization where previously repetitive serial operations 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 about loops that the compiler has parallelized.

Addressing Multithreaded Application Issues

Using Thread Analyzer

The Oracle Solaris Studio Thread Analyzer is an advanced tool for application optimization, and it 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.

Detecting Race Conditions and Deadlocks

Data races can cause incorrect or unpredictable results, and they 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 the detection of data race and deadlock issues, 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 are returned correctly.

The 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.

```
% 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.

Chapter 6 Migrating the Device Drivers and Kernel

Device driver software is tightly coupled to the operating system it is running on. In order to develop a device driver, you need to have a thorough knowledge of the operating system architecture and internals. While writing the device drivers, the operating system (kernel) interfaces that your driver will use to access the hardware must be well studied. In Oracle Solaris, the set of driver interfaces that device drivers must use while communicating with the operating system and vice versa are called the Device Driver Interfaces (DDI) and Driver-Kernel Interfaces (DKI), and they are well documented and published on Oracle's Website.

A list of data structures that can be used to share the information between device drivers and kernel on Oracle Solaris 11 is available at http://docs.oracle.com/cd/E23824_01/html/821-1478/index.html.

Entry-point routines, their syntax, specification and return values are also documented:

- Kernel to the device driver interfaces: http://docs.oracle.com/cd/E23824_01/html/821-1476/index.html
- Device driver to kernel interfaces:http://docs.oracle.com/cd/E23824_01/html/821-1477/index.html

The DDI and DKI are intended to standardize and document all interfaces between device drivers and the rest of the kernel.

Instead of going into details about DDI and DKI, this document will describe various issues, precautions, and platform-specific differences that developers must consider during migration from RHEL to Oracle Solaris 11.

Migrating Kernel Modules and Device Drivers

Platform-specific differences in system administration commands for device drivers and kernel modules are shown in Table 6-1. Entry points and implementation differences are shown in Table 6.2.

TABLE 6-1. INFRASTRUCTURE DIFFERENCES

RHEL	ORACLE SOLARIS 11
In Linux, kernel modules and device drivers are treated same. Both of them can be added using `modprobe` command	In Oracle Solaris, different sets of commands needs to be used to load kernel modules and device driver. • Kernel modules are loaded using `modload` command. • Device drivers can be added using `add_drv` command

TABLE 6-2. ENTRY POINTS AND IMPLEMENTATION DIFFERENCES

RHEL	ORACLE SOLARIS 11
The kernel module daemon `kmod` execs `modprobe`.	The kernel searches the linked list of `modctl` structures looking for a match to the module name (`mod_modname`).
`modprobe` looks through the file `/lib/modules/version/modules.dep` to see if other modules must be loaded before the requested module may be loaded.	If a match is found, return the address of the existing structure; otherwise, create a new one. Add a new `modctl` structure to the linked list.
`insmod` is fired to load prerequisite modules.	Dependencies are defined in the module code and maintained in the `modctl` structure by pointers in `mod_requisites` (other modules this module depends on) and `mod_dependents` (modules that depend on this one).
The symbol table is retrieved via `query_module()` system call.	The `_info()` function returns information about a loadable module. Within this function, the call `mod_info()` has to be made.
`insmod` links prerequisite modules	
The `create_module()` syscall is invoked next, passing the module's name and its final size.	Enter the kernel runtime linker, `krtld`, to create address space segments and bindings, and load the object into memory.
	Allocate the module `structure ()`.
	Allocate space for the module's symbols in the kernel's `kobj_map` resource map.
	Loop through the segments of the module being loaded, and allocate and map space for text and data.
	Load the kernel object into memory, linking the object's segments into the appropriate kernel address space segments.
	Set the `mod_loaded` bit in the module's `modctl` structure, and increment the `mod_loadcnt`.
`insmod` links the module to the kernel.	Create a link to the module's `mod_linkage` structure.
`insmod` calls the `init_module()` system call.	Execute the module's `mod_install` function indirectly by looking up the module `_init()` routine and calling it.
To delete a module, the `delete_module()` system call is made.	`_fini()` prepares a loadable module for unloading.
	Within `_fini()`, a call to `mod_remove()` has to be placed. It is also wise to catch the return values in order to report errors while unloading the module.

Device Driver Interface and Driver-Kernel Interface

The following table lists the differences in data structures used by kernel modules on the two platforms.

TABLE 6-3. DATA STRUCTURE AND FUNCTION CALLS

RHEL	ORACLE SOLARIS 11

`query_module()` in Linux is used to query the kernel for various bits pertaining to modules.	The `_info()` function returns information about a loadable module. Within this function, the call `mod_info()` has to be made.
	`#include <sys/modctl.h>`
	`int mod_info(struct modlinkage *modlinkage, struct modinfo *modinfo);`
	`int _info(struct modinfo *modinfop)`
	`{`
	`return (mod_info(&modlinkage, modinfop));`
	`}`

`create_module()` in Linux attempts to create a loadable module entry and reserve the kernel memory that will be needed to hold the module.	The primary data structures used in support of module loading are the `modctl` (module control) structure and module structure. You can find the structure definitions for `modctl` and module in `/usr/include/sys/modctl.h` and `/usr/include/sys/kobj.h`, respectively.
	`#include <sys/modctl.h>`
	`int mod_install(struct modlinkage *modlinkage);`
	
	`int _init(void)`
	`{`
	`int i;`
	
	`if ((i = mod_install(&modlinkage)) != 0)`
	`cmn_err(CE_NOTE,"Could not install module\n");`
	`else`
	`cmn_err(CE_NOTE,"flkm: successfully installed");`
	
	`return i;`
	`}`

`init_module()` in Linux initializes a loadable module entry.	In Oracle Solaris, `_init()` initializes a loadable module. Within an `_init()` call, you need to call another function called `mod_install()` that takes the `modlinkage struct` as an argument.

`delete_module()` in Linux deletes a loadable module.	`mod_remove()` is used to remove module in Oracle Solaris. `mod_remove()` needs to be called from within `_fini()`.
	
	`#include <sys/modctl.h>`
	`int mod_remove(struct modlinkage *modlinkage);`
	
	`int _fini(void)`
	`{`
	`int i;`
	
	`if ((i = mod_remove(&modlinkage)) != 0)`
	`cmn_err(CE_NOTE,"Could not remove module\n");`
	`else`
	`cmn_err(CE_NOTE,"flkm: successfully removed");`
	
	`return i;`
	`}`

Necessary Compiler Options, Linker Options, and Linked Libraries

TABLE 6-4. COMPILATION STEPS AND MAKEFILE CHANGES

RHEL	ORACLE SOLARIS 11
A typical kernel module compilation command will look as shown below: `gcc -O2 -DMODULE -D__KERNEL__ -W -Wall -Wstrict-prototypes -Wmissing-prototypes -isystem /lib/modules/`uname -r`/build/include -c flkm.c -o flkm`	Compiling a module is very simple. All you need to set are some definitions that tell the included code this will be a kernel module and not a normal executable. You should always link your module's object file with the `-r` option; otherwise, the module will not load, because the kernel module linker will not be able to link the module. `gcc -D_KERNEL -DSVR4 -DSOL2 -O2 -c flkm.c ld -o flkm -r flkm.o`

Best Practices for Porting Device Drivers and Kernel Modules

The porting of device drivers from one platform to the other might not be an easy job, because of one or more of the following reasons:

- To enhance functionality, in newer versions of an operating system, new members get added to the kernel data structures accessed by drivers, or the size of existing members gets redefined.
- The calling or return syntax of kernel functions gets changed.
- Driver developers do not use existing kernel functions when they are available, but instead they rely more on undocumented side effects. The issue arises particularly when these functions and their behavior are not maintained in the next release of an operating system.
- Architecture-specific code is scattered throughout the driver, when it should have been isolated. Isolating the portable code from nonportable helps reduce porting issues.

The Oracle Solaris DDI and DKI enable you to write platform-independent device drivers, that is, drivers which work on any machine that runs Oracle Solaris. These binary-compatible drivers enable you to more easily integrate any third-party hardware and software into any machine that runs Oracle Solaris. The DDI and DKI are architecture-independent, which enables the same driver to work across a diverse set of machine architectures (SPARC as well as x86).

The primary goal of the DDI and DKI is to facilitate both source and binary portability across successive releases of the operating systems on a particular machine. Based on your hardware and software support metric requirements, to achieve the goal of source and binary compatibility, the functions, routines, and structures specified in the DDI and DKI must be used taking the following facts into consideration.

- The DDI and DKI consist of several sections:
  - Oracle Solaris SPARC-specific DDI: These interfaces are specific to the SPARC processor and might not be available on other processors supported by Oracle Solaris.
  - Oracle Solaris x86-specific DDI: These interfaces are specific to the x86 processor and might not be available on other processors supported by Oracle Solaris.
  - Oracle Solaris DDI: These interfaces are specific to Oracle Solaris.
  - DKI-only: These interfaces are part of System V Release 4 and might not be supported in future releases of System V. There are only two interfaces in this class: `segmap` and `hat_getkpfnum`.
  - DDI/DKI Architecture Independent: These interfaces are supported on all implementations of System V Release 4.
- Drivers that use only kernel facilities that are part of the DDI/DKI are known as DDI/DKI-compliant device drivers.

Chapter 7 Security

Security Interfaces for Developers

For data centers hosting enterprise applications, security is of utmost importance. In most enterprises, security is considered one of the prime factors for making platform decisions. To build a secure environment, rather than looking at security as set of commands and features available in operating systems, it is necessary that the security features be designed into the core of an operating system. Built over decades, Oracle Solaris provides unmatched enterprise-class features you can depend on to protect your applications. Oracle Solaris 11 combines multiple security technologies—from networking, cryptographic capabilities, and trusted extensions to process and user rights as well as unmatched monitoring and auditing capabilities.

Irrespective of the hosting operating system (Linux or Oracle Solaris), to build a secure system, security administrators follow simple rules:

- Ensure physical security.
- Deploy stringent access controls.
- Simplify administration.
- Delegate appropriate (minimal) privileges.
- Use Oracle Solaris Trusted Extensions.
- Do minimal installs.
- Ensure strong defenses.

Physical Security

The first and foremost layer of security you need to take into account is the physical security of the computer systems. How much physical security you need on a system is dependent on your situation and other logistics such as using shared labs, shared systems, deployments in a virtualized environment on a larger server, and so on. The various measures that need to be taken to ensure security involve things such as securing direct physical access to a machine and the security of connected peripheral and devices, as well as restricting access to system details such as BIOS passwords, screen-saver settings, password policies for console login, and so on.

Delegate Minimal Privileges—Only as Appropriate

A privilege is a discrete right that can be granted to an application. With a privilege, a process can perform an operation that would otherwise be prohibited by the operating system. Legacy UNIX systems follow a superuser-based model. So a typical UNIX application will have checks for UID (0/`root`) to test for the availability of specific privileges. This has drastically changed in Oracle Solaris. Oracle Solaris now implements a *least privilege model*, which gives a specified process only a subset of the superuser powers, not full access to all privileges.

The least privilege model includes nearly 50 fine-grained privileges as well as the basic privilege set. With the help of process privileges, system administrators now can delegate limited permission to users to override system security instead of giving users complete `root` access. On Oracle Solaris, a user does not have to become superuser to use a privileged application. In Oracle Solaris, privileges are enforced at the kernel level.

Authorizations can also be used to give additional permissions. An authorization is permission for performing a class of actions that are otherwise prohibited by the security policy. An authorization can be assigned to a role or a user. Authorizations are enforced at the user level.

On Oracle Solaris, when an application runs, privileges can be turned on or turned off programmatically. By default, when a new program is started, that program can potentially use all of the inheritable privileges of the parent process.

Appendix A provides a list of Oracle Solaris security privileges and their definitions.

Oracle Solaris Trusted Extensions

If you need a highly stringent system security, using the Oracle Solaris Trusted Extensions feature is a good choice. Trusted Extensions software provides special security features and extends the capabilities of Oracle Solaris to enable you to define and implement a security policy on an Oracle Solaris system. You can get more information about Trusted Extensions at http://docs.oracle.com/cd/E18752_01/html/819-0868.

Ensure Strong Defenses

The internet comprises hundreds of thousands of networks that are interconnected without boundaries. In today's business environment, you will hardly find any standalone servers catering to meaningful business needs. Network sharing and data sharing have become an essential part of any enterprise system deployment. With the advancement of the internet and interconnected networks, network security has become essential. Part of almost every organizational network is accessible from other computers across the world and is, therefore, potentially vulnerable to threats from individuals who might not have any physical access.

Oracle Solaris provides a network security architecture that is based on standard industry interfaces. As security technologies evolve, application developers do not have to modify their code if they use the standardized interfaces. The Oracle Solaris network security architecture works with standard industry interfaces, such as PAM, GSS-API, SASL, PKCS#11. Using the network security architecture eliminates the need for developers to write, maintain, and optimize cryptographic algorithms. Optimized cryptographic mechanisms are provided in Oracle Solaris 11 as part of the operating system.

The cryptographic framework is the backbone of cryptographic services in Oracle Solaris. The framework provides standard PKCS #11 interfaces to accommodate consumers and providers of cryptographic services. The framework has two parts:

- A user cryptographic framework for user-level applications

- A kernel cryptographic framework for kernel-level modules

Note that consumers that are connected to the framework need not have any specific knowledge of the installed cryptographic mechanisms. Similarly, providers can be plugged into the framework and can support different types of consumers. You do not have to write consumer-specific code in the providers.

Encryption Algorithms, Mechanisms, and Their Mapping

The Oracle Solaris cryptographic framework provides cryptographic services to users and applications through individual commands, user-level and kernel-level frameworks, and user and kernel programming interfaces. The cryptographic framework provides these cryptographic services to applications and kernel modules in a manner seamless to the end user. End users can also make use of direct cryptographic services, such as encryption and decryption of files.

With the availability of an onboard crypto accelerator on Oracle's SPARC T4 chip, all applications written to use the cryptographic framework can take advantage of this new hardware crypto accelerator without doing any additional work.

The following table lists the differences in function calls on the two platforms.

TABLE 7-1. API AND IMPLEMENTATION DIFFERENCES

RHEL	ORACLE SOLARIS 11
MD4	
`unsigned char *MD4(const unsigned char *d, unsigned long n, unsigned char *md);`	
`int MD4_Init(MD4_CTX *c);`	`void MD4Init(MD4_CTX *context);`
`int MD4_Update(MD4_CTX *c, const void *data, unsigned long len);`	`void MD4Update(MD4_CTX *context, unsigned char *input, unsigned int inlen);`
`int MD4_Final(unsigned char *md, MD4_CTX *c);`	`void MD4Final(unsigned char *output, MD4_CTX *context);`
MD5	
`unsigned char *MD5(const unsigned char *d, unsigned long n, unsigned char *md);`	`void md5_calc(unsigned char *output, unsigned char *input, unsigned int inlen);`
`int MD5_Init(MD5_CTX *c);`	`void MD5Init(MD5_CTX *context);`
`int MD5_Update(MD5_CTX *c, const void *data, unsigned long len);`	`void MD5Update(MD5_CTX *context, unsigned char *input, unsigned int inlen);`

TABLE 7-1. API AND IMPLEMENTATION DIFFERENCES

RHEL	ORACLE SOLARIS 11
`int MD5_Final(unsigned char *md, MD5_CTX *c);`	`void MD5Final(unsigned char *output, MD5_CTX *context);`
SHA1	
`unsigned char *SHA1(const unsigned char *d, unsigned long n, unsigned char *md);`	
`int SHA1_Init(SHA_CTX *c);`	`void SHA1Init(SHA1_CTX *context);`
`int SHA1_Update(SHA_CTX *c, const void *data, unsigned long len);`	`void SHA1Update(SHA1_CTX *context, unsigned char *input, unsigned int inlen);`
`int SHA1_Final(unsigned char *md, SHA_CTX *c);`	`void SHA1Final(unsigned char *output, SHA1_CTX *context);`
RSA	
`int RSA_get_ex_new_index(long argl, char *argp, int (*new_func)(), int (*dup_func)(), void (*free_func)());`	`int RSA_get_ex_new_index(long argl, void *argp, CRYPTO_EX_new *new_func, CRYPTO_EX_dup *dup_func, CRYPTO_EX_free *free_func);`
`int RSA_set_ex_data(RSA *r,int idx,char *arg);`	`int RSA_set_ex_data(RSA *r, int idx, void *arg);`
`char *RSA_get_ex_data(RSA *r, int idx);`	`void *RSA_get_ex_data(RSA *r, int idx);`
	`typedef int CRYPTO_EX_new(void *parent, void *ptr, CRYPTO_EX_DATA *ad, int idx, long argl, void *argp);`
	`typedef void CRYPTO_EX_free(void *parent, void *ptr, CRYPTO_EX_DATA *ad, int idx, long argl, void *argp);`
	`typedef int CRYPTO_EX_dup(CRYPTO_EX_DATA *to, CRYPTO_EX_DATA *from, void *from_d, int idx, long argl, void *argp);`
`RSA *RSA_new_method(ENGINE *engine);`	`RSA *RSA_new_method(RSA_METHOD *method);`
`const RSA_METHOD *RSA_get_default_method(void);`	`RSA_METHOD *RSA_get_default_method(void);`
THE FOLLOWING IMPLEMENTATIONS ARE THE SAME ON BOTH THE PLATFORMS:	
DES	

TABLE 7-1. API AND IMPLEMENTATION DIFFERENCES

RHEL	ORACLE SOLARIS 11
Blowfish	
Arcfour or RC4	

Table 7-2 shows the differences in names and locations of header files.

TABLE 7-2. DIFFERENCES IN HEADER FILES

RHEL	ORACLE SOLARIS 11
MD4	
`#include <openssl/md4.h>`	`#include <md4.h>`
MD5	
`#include <openssl/md5.h>`	`#include <md5.h>`
SHA1	
`#include <openssl/sha.h>`	`#include <sha1.h>`

Using Hardware Accelerators and System-Provided Interfaces

The SPARC T4 processor from Oracle is designed with security as a focus and has crypto instruction accelerators integrated directly into each processor core. Integrating encryption capabilities directly inside the instruction pipeline of the processor eliminates the performance and cost barriers associated with secure computing.

The Oracle Solaris cryptographic framework can fully leverage hardware-assisted cryptographic acceleration provided by Oracle's SPARC T-Series processors as well as other third-party PKCS#11-based hardware security modules transparently. This and the following are examples of benefits provided by the integrated hardware and software stack provided by Oracle:

On Oracle Solaris 11, both Java and non-Java applications can delegate SSL/TLS and WS-Security tasks involved with compute-intensive public key encryption, bulk encryption, and digest operations to hardware via a Java PKCS#11 provider.

- The Oracle Solaris OpenSSL pkcs11 engine will automatically leverage hardware-assisted cryptographic acceleration support provided by SPARC T-Series processors and Intel Westmere (AES-NI) processors.

- The hardware-assisted encryption can be used to enhance Oracle Database performance with Transparent Data Encryption (TDE) for column-level and table-space encryption.

For more information on hardware-assisted encryption and how to benefit from it without making fundamental changes to application code, see the "Oracle Advanced Security Transparent Data Encryption Best Practices" white paper at http://www.oracle.com/technetwork/database/focus-areas/security/twp-transparent-data-encryption-bes-130696.pdf.

Chapter 8 Runtime Environment

Though RHEL and Oracle Solaris are UNIX-based operating systems, there are some fundamental differences in their runtime environments. One such difference which might affect application is the privilege models on the two systems. Oracle Solaris uses the new *least privilege model*, which gives a specified process only a subset of superuser powers, not full access to all privileges.

User and Process Privileges and Permissions

Legacy UNIX systems follow a superuser-based model. Hence, either an application is designed to run as a "normal user" and have very limited privileges or it will have to run directly as the `root` user and have almost all privileges. Typically, an application ported from such legacy operating systems might have checks for UID (0/`root`) to test for the availability of specific privileges. On the other hand, on Oracle Solaris, which uses the least privilege model, a user can have some of the 50 fine-grained privileges.

The Oracle Solaris least privilege model conveniently enables normal users to do things such as bind to lower-numbered ports, start daemon processes, and so on. On the other hand, it also protects the system against programs that previously ran with full `root` privileges because they needed very limited access, for example, mounting a file system. Hence, `setuid` root binaries and daemons that run with full `root` privileges on RHEL are rarely necessary under the Oracle Solaris least privilege model.

While migrating an application from RHEL to Oracle Solaris per security best practices, try running the application as a normal least privileged user, and then add necessary privileges to the process. In order to determine which privilege is missing for a given application, use the debugging functionality of `ppriv` in the shell, for example:

```
User1>/usr/bin/ppriv -D $$                // Enable ppriv debugging

User1>./my_net_application  arg1   //Execute the application
     my_net_application[2885]: missing privilege "net_icmpaccess" (euid = 2002,
            syscall = 230) for "devpolicy" needed at so_socket+0xa4
     my_net_application: icmp socket: Permission denied

User1>ppriv -N $$                 //Disable ppriv debugging
```

From this, it is now evident that the `PRIV_NET_ICMPACCESS` privilege is not available for UID 2002. Once granted, my_net_application will start running successfully.

The role-based access control (RBAC) facility in Oracle Solaris 11 gives you the opportunity to deliver fine-grained security in new and modified applications. RBAC is an alternative to the all-or-nothing security model of traditional superuser-based systems. With RBAC, an administrator can assign privileged functions to specific user accounts (or special accounts called *roles*). RBAC can also be used

with the least privilege model to more securely run daemons such as `httpd`, which need to bind to privileged ports. Many such programs do not actually need `root` access for anything other than listening on a port below 1024, so granting the role to a user that runs the process `net_privaddr` would remove the need for ever running the process with EUID 0.

Oracle Solaris RBAC configuration is controlled through four main files:

`/etc/security/exec_attr`, `/etc/security/prof_attr`, `/etc/security/auth_attr`, and `/etc/user_attr`.

- `exec_attr` specifies the execution attributes associated with profiles. This generally includes the user and group IDs, commands, and default/limit privileges.
- `prof_attr` specifies a collection of execution profile names, descriptions, and other attributes.
- `auth_attr` specifies authorization definitions and descriptions.
- `user_attr` specifies user and role definitions along with their assigned authorizations, profiles, and projects.

For security reasons, by default, only `root` has the privilege to use DTrace. To allow a group of users to use DTrace, a system administrator would either create a role that had access to the DTrace privileges or assign the privilege directly to the user.

To create a `support` role and grant appropriate privileges, use the following steps:

```
#roleadd -u 201 -d /export/home/support -P "Process Management" support
#rolemod -K defaultpriv=basic,dtrace_kernel,dtrace_proc,dtrace_user support
```

Now the users with the role `support` can use `su` to access `support` (after providing the appropriate password) and run the necessary DTrace commands.

Instead of adding roles and making the users access the role via `su`, a system administrator can also directly assign privileges to a user:

```
#usermod -K defaultpriv=basic,dtrace_kernel,dtrace_proc,dtrace_user USER1
```

Note: The user must be logged out in order for the command above to succeed.

Privilege awareness is a mechanism that allows legacy applications to retain full compatibility with the traditional *full privilege model*. Each process also has a privilege awareness state that can be set to *privilege aware* (PA) or *not-privilege aware* (NPA). Legacy applications that are NPA will appear to be granted all privileges (in the set L) if EUID, RUID, or SUID are 0 (`root`). Note that when a process calls `exec`, the kernel tries to relinquish privilege awareness.

In Oracle Solaris, although most of the normal `setuid`/`setgid` executables have been rewritten to be privilege aware (PA), they still have their `setuid`/`setgid` flags set. This is necessary for the program to first gain the appropriate `root` privilege and then drop the unnecessary ones. However, this would require the program to be fully privilege aware and some `setuid`/`setgid` programs might not have been ported yet.

Resource Controls and Runtime Limits

When moving from RHEL to Oracle Solaris 11, one reason an application might face problems is a difference in the *resource limits* on the two platforms. The following information can be used to determine if any changes are required on your system.

Resource Limits on RHEL

On typical RHEL, the `ipcs -l` command will show the following output. Comments have been added following the `//` to show what the parameter names are.

```
# ipcs -l
------ Shared Memory Limits --------
max number of segments = 4096                    // SHMMNI
max seg size (kbytes) = 67108864                 // SHMMAX
max total shared memory (kbytes) = 17179869184 // SHMALL
min seg size (bytes) = 1
------ Semaphore Limits --------
max number of arrays = 128                        // SEMMNI
max semaphores per array = 250                    // SEMMSL
max semaphores system wide = 32000         // SEMMNS
max ops per semop call = 32                       // SEMOPM
semaphore max value = 32767
------ Messages: Limits --------
max queues system wide = 15493                            // MSGMNI
max size of message (bytes) = 65536                       // MSGMAX
default max size of queue (bytes) = 65536         // MSGMNB
```

To modify these kernel parameters on RHEL, edit the `/etc/sysctl.conf` file. The following lines are examples of what should be placed into the file:

```
kernel.sem=250 256000 32 1024
#Example shmmax for a 64-bit system
kernel.shmmax=1073741824
#Example shmall for 90 percent of 16 GB memory
kernel.shmall=3774873
kernel.msgmax=65535
kernel.msgmnb=65535
```

You need to run `sysctl` with the `-p` parameter, to load in `sysctl` settings from the default file `/etc/sysctl.conf`.

```
#sysctl –p
```

On RHEL, after reboot, the `rc.sysinit` initialization script reads the `/etc/sysctl.conf` file automatically.

Resource Limits on Oracle Solaris 11

On Oracle Solaris 11, the default resource limits are set very high compared to previous Oracle Solaris releases. The `/etc/system` file provides a static mechanism for adjusting the values of kernel parameters. Values specified in this file are read at boot time and are applied. Any changes that are made to the file are not applied to the operating system until the system is rebooted.

You can use the `sysdef` command to see the values of system-V IPC settings, STREAMS tunables, and process resource limits.

The `/etc/project` file contains settings for multiple resource controls for each project as well as multiple threshold values for control. Use `proj*` (`projadd`, `projdel`, `projmod`, `projects`) commands to view and modify various values. For more details on Oracle Solaris resource controls, refer to http://docs.oracle.com/cd/E23824_01/html/821-1460/rmctrls-1.html.

Migrating Scripts

UNIX applications and development environments commonly use scripts. This includes shell scripts (such as those written for the K shell), Perl scripts, and scripts written in other languages. If the scripting language used on RHEL is also supported in Oracle Solaris 11, migrating the scripts is a straightforward exercise.

When migrating from RHEL to Oracle Solaris 11, particularly in the context of migrating legacy scripts, the availability of popular GNU commands, along with corresponding similar command-line options, is critical in determining the migration effort. Fortunately, for many of these utilities, the legacy Oracle Solaris versions as well as the GNU versions are readily available on the Oracle Solaris system if you install some of the optional packages on top of the default installation.

Equivalent Commands, Options, and Alternatives on Oracle Solaris 11

Table 8-1 provides a summary of key command differences between RHEL and Oracle Solaris 11. You can get detailed information on these and other commands in the Oracle Solaris 11 man pages.

TABLE 8-1. COMMAND DIFFERENCES

RHEL	LEGACY COMMANDS ON ORACLE SOLARIS 11	INCOMPATIBLE OPTIONS WITH LEGACY COMMAND	EQUIVALENT GNU UTILITY ON ORACLE SOLARIS 11	PACKAGE NAME ON ORACLE SOLARIS 11
`/usr/bin/ar`	`/usr/bin/ar`	`-D -l -f -N -o -P`	`/usr/bin/gar: link to /usr/gnu/bin/ar`	`text/doctools`
`/bin/chgrp`	`/usr/bin/chgrp`	`-c --dereference --no-preserve-root --preserve-root --reference`	`/usr/bin/gchgrp: link to /usr/gnu/bin/chgrp`	`system/xopen/xcu4`

TABLE 8-1. COMMAND DIFFERENCES

RHEL	LEGACY COMMANDS ON ORACLE SOLARIS 11	INCOMPATIBLE OPTIONS WITH LEGACY COMMAND	EQUIVALENT GNU UTILITY ON ORACLE SOLARIS 11	PACKAGE NAME ON ORACLE SOLARIS 11
/bin/chown	/usr/bin/chown	--dereference --from --no-preserve-root --preserve-root	/usr/bin/gchown: link to /usr/gnu/bin/chown	system/xopen/xcu4
/usr/bin/cmp	/usr/bin/cmp	-b	/usr/bin/gcmp: link to /usr/gnu/bin/cmp	text/gnu-diffutils
/usr/bin/comm	/usr/bin/comm	--check-order --nocheck-order --output-delimiter	/usr/bin/gcomm: link to /usr/gnu/bin/comm	system/core-os
/bin/cp	/usr/bin/cp	-a --backup -b --copy-contents -d -l --preserve -c --no-preserve --parents --reflink --remove-destination --sparse --strip-trailing-slashes -s -S -t -u -v -x -Z	/usr/bin/gcp: link to /usr/gnu/bin/cp	system/xopen/xcu4
/usr/bin/csplit	/usr/bin/csplit	-b -z	/usr/bin/gcsplit: link to /usr/gnu/bin/csplit	system/core-os
/bin/cut	/usr/bin/cut	--complement --output-delimiter	/usr/bin/gcut: link to /usr/gnu/bin/cut	system/core-os
/bin/date	/usr/bin/date	-d -f -r -R --rfc-3339 -s	/usr/bin/gdate: link to /usr/gnu/bin/date	runtime/ruby-18
/bin/df	/usr/bin/df link to /usr/sbin/df	-B -H -i -T -x	/usr/bin/gdf: link to /usr/gnu/bin/df	system/core-os
/usr/bin/diff3	/usr/bin/diff3	-A -m -L -i -a --diff-program	/usr/bin/gdiff3: link to /usr/gnu/bin/diff3	text/gnu-diffutils
/usr/bin/du	/usr/bin/du	--apparent-size -b -c --files0-from --si -l -P -0 -S -X --exclude --max-depth --time --time-style	/usr/bin/gdu: link to /usr/gnu/bin/du	system/xopen/xcu4
/bin/echo	/usr/bin/echo	-n -e -E	/usr/bin/gecho: link to /usr/gnu/bin/echo	runtime/ruby-18

TABLE 8-1. COMMAND DIFFERENCES

RHEL	LEGACY COMMANDS ON ORACLE SOLARIS 11	INCOMPATIBLE OPTIONS WITH LEGACY COMMAND	EQUIVALENT GNU UTILITY ON ORACLE SOLARIS 11	PACKAGE NAME ON ORACLE SOLARIS 11
/bin/env	/usr/bin/env	-0 -u	/usr/bin/genv: link to /usr/gnu/bin/env	runtime/ruby-18
/usr/bin/expand	/usr/bin/expand	-i	/usr/bin/gexpand: link to /usr/gnu/bin/expand	system/core-os
/usr/bin/fmt	/usr/bin/fmt	-p -t -u	/usr/bin/gfmt: link to /usr/gnu/bin/fmt	system/core-os
/usr/bin/fold	/usr/bin/fold	-c	/usr/bin/gfold: link to /usr/gnu/bin/fold	image/editor/bitmap
/bin/grep	/usr/bin/grep	-G -P -y --color -L -m -o --label -u -Z -A -C -a --binary-files -d --exclude --exclude-from --exclude-dir -I --include -R --line-buffered --mmap -U -z	/usr/bin/ggrep: link to /usr/gnu/bin/grep	developer/quilt
/usr/bin/id	/usr/bin/id	-Z	/usr/bin/gid: link to /usr/gnu/bin/id	system/fru-id
/usr/bin/join	/usr/bin/join	--check-order --nocheck-order	/usr/bin/gjoin: link to /usr/gnu/bin/join	network/chat/ircii
/bin/kill	/usr/bin/kill	-1 -a -p	/usr/bin/gkill: link to /usr/gnu/bin/kill	runtime/ruby-18
/bin/ln	/usr/bin/ln	--backup -b -d -i -L -S -t	/usr/bin/gln: link to /usr/gnu/bin/ln	file/gnu-coreutils

TABLE 8-1. COMMAND DIFFERENCES

RHEL	LEGACY COMMANDS ON ORACLE SOLARIS 11	INCOMPATIBLE OPTIONS WITH LEGACY COMMAND	EQUIVALENT GNU UTILITY ON ORACLE SOLARIS 11	PACKAGE NAME ON ORACLE SOLARIS 11
/bin/ls	/usr/bin/ls	--author --format --group-directories-first -G --dereference-command-line-symlink-to-dir --indicator-style -I -N --show-control-chars -Q --quoting-style --sort -X --lcontext -Z --scontext	/usr/bin/gls: link to /usr/gnu/bin/ls	driver/audio/audiols
/usr/bin/m4	/usr/bin/m4	-i -P -Q --warn-macro-sequence -I -g -G -L -F -R -d --debugfile -l -t	/usr/bin/gm4: link to /usr/gnu/bin/m4	developer/macro/gnu-m4
/bin/mkdir	/usr/bin/mkdir	-v -Z	/usr/bin/gmkdir: link to /usr/gnu/bin/mkdir	system/core-os
/bin/mktemp	/usr/bin/mktemp	--suffix --tmpdir	/usr/bin/gmktemp: link to /usr/gnu/bin/mktemp	system/core-os
/usr/bin/msgfmt	/usr/bin/msgfmt	-j --java2 --csharp --csharp-resources --tcl --qt -r -l -d -P --stringtable-input --check-format --check-header --check-domain -C --check-accelerators -a --no-hash --statistics	/usr/bin/gmsgfmt: link to /usr/gnu/bin/msgfmt	text/locale
/bin/mv	/usr/bin/mv	--backup -b --strip-trailing-slashes -S -t -u -v	/usr/bin/gmv: link to /usr/gnu/bin/mv	x11/keyboard/data-xkb
/usr/bin/od	/usr/bin/od	-w --traditional	/usr/bin/god: link to /usr/gnu/bin/od	system/xopen/xcu4

TABLE 8-1. COMMAND DIFFERENCES

RHEL	LEGACY COMMANDS ON ORACLE SOLARIS 11	INCOMPATIBLE OPTIONS WITH LEGACY COMMAND	EQUIVALENT GNU UTILITY ON ORACLE SOLARIS 11	PACKAGE NAME ON ORACLE SOLARIS 11
/usr/bin/pathchk	/usr/bin/pathchk	-P --portability	/usr/bin/gpathchk: link to /usr/gnu/bin/pathchk	system/core-os
/usr/bin/pr	/usr/bin/pr	-J -N -v -W	/usr/bin/gpr: link to /usr/gnu/bin/pr	text/groff/groff-core
/usr/bin/refer	/usr/bin/refer	-C -P -S -f -i -t -R	/usr/bin/grefer: link to /usr/gnu/bin/refer	text/doctools
/bin/rm	/usr/bin/rm	-I --interactive --one-file-system --no-preserve-root --preserve-root	/usr/bin/grm: link to /usr/gnu/bin/rm	system/xopen/xcu4
/usr/bin/sdiff	/usr/bin/sdiff	-i -b -W -I --strip-trailing-cr -a -t -d -H --diff-program	/usr/bin/gsdiff: link to /usr/gnu/bin/sdiff	text/gnu-diffutils
/usr/bin/size	/usr/bin/size	-A --format -d --radix --common -t --target	/usr/bin/gsize: link to /usr/gnu/bin/size	developer/gnu-binutils
/bin/sort	/usr/bin/sort	-g -h -R --random-source --sort -V --batch-size -C --compress-program --files0-from -s	/usr/bin/gsort: link to /usr/gnu/bin/sort	system/xopen/xcu4
/usr/bin/split	/usr/bin/split	-C -d --verbose	/usr/bin/gsplit: link to /usr/gnu/bin/split	system/core-os
/usr/bin/strings	/usr/bin/strings	-f --bytes --radix --encoding --target	/usr/bin/gstrings: link to /usr/gnu/bin/strings	developer/gnu-binutils

TABLE 8-1. COMMAND DIFFERENCES

RHEL	LEGACY COMMANDS ON ORACLE SOLARIS 11	INCOMPATIBLE OPTIONS WITH LEGACY COMMAND	EQUIVALENT GNU UTILITY ON ORACLE SOLARIS 11	PACKAGE NAME ON ORACLE SOLARIS 11
/usr/bin/strip	/usr/bin/strip	-F --target --info -I --input-target -O --output-target -R --remove-section --strip-all -g -S -d --strip-debug --strip-unneeded -K --keep-symbol -N --strip-symbol -o -p --preserve-dates -w ---discard-all -X --keep-file-symbols --only-keep-debug -v	/usr/bin/gstrip: link to /usr/gnu/bin/strip	developer/gnu-binutils
/bin/stty	/usr/bin/stty	-F	/usr/bin/gstty: link to /usr/gnu/bin/stty	system/xopen/xcu4
/usr/bin/tail	/usr/bin/tail	-F --max-unchanged-stats --pid --retry	/usr/bin/gtail: link to /usr/gnu/bin/tail	system/xopen/xcu4
/bin/tar	/usr/bin/tar link to /usr/sbin/tar	Several parameters are not available	/usr/bin/gtar: link to /usr/gnu/bin/tar	runtime/perl-512
/bin/touch	/usr/bin/touch	-h --time	/usr/bin/gtouch: link to /usr/gnu/bin/touch	file/gnu-coreutils
/usr/bin/tput	/usr/bin/tput	-V	/usr/bin/gtput: link to /usr/gnu/bin/tput	system/core-os
/usr/bin/tr	/usr/bin/tr	-t	/usr/bin/gtr: link to /usr/gnu/bin/tr	text/groff
/bin/uname	/usr/bin/uname link to /usr/sbin/uname	-o	/usr/bin/guname: link to /usr/gnu/bin/uname	system/core-os
/usr/bin/uniq	/usr/bin/uniq	-z	/usr/bin/guniq: link to /usr/gnu/bin/uniq	system/core-os
/usr/bin/wc	/usr/bin/wc	--files0-from	/usr/bin/gwc: link to /usr/gnu/bin/wc	system/kernel
/usr/bin/who	/usr/bin/who	--lookup	/usr/bin/gwho: link to /usr/gnu/bin/who	system/dtrace/dtrace-toolkit

TABLE 8-1. COMMAND DIFFERENCES

RHEL	LEGACY COMMANDS ON ORACLE SOLARIS 11	INCOMPATIBLE OPTIONS WITH LEGACY COMMAND	EQUIVALENT GNU UTILITY ON ORACLE SOLARIS 11	PACKAGE NAME ON ORACLE SOLARIS 11
`/usr/bin/xargs`	`/usr/bin/xargs`	`-a -d -P`	`/usr/bin/gxargs: link to /usr/gnu/bin/xargs`	`file/gnu-findutils`
`/usr/bin/zenity`	`/usr/bin/zenity`	`--no-wrap`		`gnome/zenity`
`/bin/more`	`/usr/bin/more`	`-num`		`system/xopen/xcu4`
`/usr/bin/atrm`	`/usr/bin/atrm`	`-d`	`-d` is not a necessary argument; instead of `-l`, `atrm` command can be used.	`system/core-os`
`/usr/bin/atq`	`/usr/bin/atq`	`-l -d`	`-l` is not a necessary argument; instead of `-d`, `atrm` command can be used.	`system/core-os`
`/usr/bin/at`	`/usr/bin/at`	`-d -v`	Instead of `–d`, `atrm` can be used; instead of `at -v` in Linux, `at -l` can be used in Oracle Solaris to find execution time.	`system/xopen/xcu4`
`/usr/bin/rup`	`/usr/bin/rup`	`-d -s`	Instead of the `-d` and `-s` options in Linux, a combination of options for the `date` command can be used.	`network/legacy-remote-utilities`

Managing Services and Daemons on RHEL

A UNIX daemon is a program that runs in the background and is independent of control from a terminal. Daemons can either be started by a process (such as a system startup script, where there is no controlling terminal) or by users at a terminal without tying up the terminal.

The `init` daemon is the system and service manager for RHEL. It is one of the first processes RHEL starts when it boots. It has a PID of 1 and is the ancestor of all processes. On RHEL, services are started and stopped through `init` scripts in the `/etc/init.d` directory. Many of these services are launched by the `init` daemon when the system is booted.

Many System V UNIX variants use scripts in the `/etc/rcN.d/` directories to control which services should be started in the various run levels. Since this model involved having multiple copies of the same script in many different directories, RHEL adopted the standard of putting all service control scripts in the `/etc/init.d/` directory and using symbolic links to these scripts in the various

`/etc/rcN.d/` directories. With this arrangement, now it is possible to have centralized commands such as `chkconfig` and `/sbin/services` manage all services from a single interface.

RHEL Service Management

In RHEL, the `/sbin/service` utility provides a consistent interface for executing the `init` scripts. The `init` scripts provide a consistent interface for managing a service by providing options to start, stop, restart, query status, and reload and to perfrom other actions on services. Since nearly all services on a server need high privileges, you need to log in as `root` to control them.

You can view the current state of all services with the `-status-all` option of the service utility:

```
#/sbin/service –status-all
abrtd (pid  1762) is running...
acpid (pid  1477) is running...
atd (pid  1817) is running...
```

The run-level information for these services (that is, the system run level at which the service will be started) can be queried and modified with the `chkconfig` utility.

For example, to see the current settings for the `crond` service, execute the following command:

```
#/sbin/chkconfig --list crond
  crond             0:off   1:off   2:on    3:on    4:on    5:on    6:off
```

This shows that the `crond` service will be automatically started at boot time for run-levels 2, 3, 4, and 5.

To set the service to not start for run levels 3 and 4, you can use the `chkconfig` utility as shown below:

```
#/sbin/chkconfig --level 34 crond off
```

To disable a specific daemon, you need to execute the command shown below:

```
#/sbin/chkconfig daemon-name off
```

Replace *`daemon-name`* with the name of the service you wish to disable.

Managing and Controlling Service Dependencies

RHEL has all service control scripts in `/etc/init.d/`, and then the run level at which each of these services is started is controlled using symbolic links to these scripts in the various `/etc/rcN.d/` directories.

Let's look at what happens when we enable `crond` only in run level 5.

We run the following command to accomplish this:

```
#/sbin/chkconfig --list crond
  crond             0:off   1:off   2:on    3:on    4:on    5:on    6:off
# /sbin/chkconfig crond --level 234 off
# /sbin/chkconfig --list crond
  crond 0:off 1:off 2:off 3:off 4:off 5:on 6:off
```

Now you can go to directories `/etc/rc2.d/`, `/etc/rc3.d/`, and `/etc/rc4.d/` to confirm that the link to the `crond` startup script really got deleted.

Let's look at a typical service startup script to understand how services are controlled:

```
##sample crond startup script##
#! /bin/sh
case "$1" in
    start)
        rh_status_q && exit 0
        $1
        ;;
    stop)
        rh_status_q || exit 0
        $1
        ;;
    restart)
        $1
        ;;
    reload)
        rh_status_q || exit 7
        $1
#### /crond ####
```

Now, based on the user inputs, the appropriate part of the script gets executed.

Oracle Solaris Service Management Facility (SMF)

Oracle Solaris Service Management Facility (SMF) provides an infrastructure that augments the traditional UNIX startup scripts, `init` run levels, and configuration files. With the availability of SMF on Oracle Solaris 11, system administrators can use simple commands to easily identify, observe, and manage services on the system.

An Oracle Solaris service is any long-lived software object with a well-defined state and start and stop controls that has a relationship to and dependencies on other services in the system. SMF can be configured to build a self-healing service infrastructure. Through SMF, failing services can be automatically restarted whenever possible, reducing the need for human intervention. If manual

intervention is required, system administrators can quickly identify the root cause of the service's failure and significantly reduce system down time.

By moving to SMF on Oracle Solaris 11, system administrators will be able to perform the following tasks through a single framework:

- Observe and manage system-wide services.
- Provide consistent configuration handling.
- Automatically restart failed services in the appropriate order of dependency.
- Identify misbehaved or failed services.
- Securely delegate administrative tasks to non-`root` users.
- Preserve compatibility with legacy services.
- Automatically configure system administration jobs such as backup, restore, and so on.

Managing Services Through SMF

Here are some of the most important SMF commands and their purpose:

- `svcs`: Reports a service's status.
- `svcadm`: Used for service management, for example, for starting, stopping, and restoring services.
- `svccfg`: Used to import, export,and modify service configurations.
- `svcprop`: Used to list the properties of a service.
- `inetadm`: Used to manage `inetd` services.SMF provides a uniform mechanism for managing system services. SMF defines a set of actions that can be invoked on a service by an administrator. These actions include commands to enable, disable, refresh, or restart a service.

For example, you can view the current state of all services with the following command:

```
#svcs -a
STATE          STIME    FMRI
legacy_run     Mar_12   lrc:/etc/rc2_d/S10lu
legacy_run     Mar_12   lrc:/etc/rc2_d/S20sysetup
legacy_run     Mar_12   lrc:/etc/rc2_d/S47pppd
online         Mar_12   svc:/system/svc/restarter:default
online         Mar_12   svc:/network/loopback:default
online         Mar_12   svc:/network/tnctl:default
:
:
```

You can start the `cron` daemon with the following command:

```
#svcadm enable cron
```

You can disable a service with the following command, where *service name* is the name of the service that you want to disable:

```
#svcadm disable service name
```

Managing Service Dependencies in Oracle Solaris

SMF provides a simple mechanism for defining the relationships among various services so that dependent services can be automatically restarted when necessary. Information necessary to manage each service is stored in the service repository. This information allows the restarter to move the service from one state to another state. Each service is managed by a service restarter, which carries out the administrative actions.

The service configuration repository provides a per-service snapshot at the time each service is successfully started so that fallback is possible. This capability helps you debug service configuration problems easily.

The fundamental unit of administration in the SMF framework is the service instance. Each SMF service has the potential to have multiple versions of it configured, as well multiple instances of the same version running on a single system.

An instance is a specific configuration of a service, for example, a Web server is a service and a specific Web server daemon that is configured to listen on port 8080 is an instance. Each instance of the Web server service can be configured to have different specifications. The Web service will have system-wide configuration settings, and each instance can choose to override specific configuration.

Migrating to SMF

Migrating services from RHEL to Oracle Solaris SMF is not a difficult task because SMF preserves compatibility with legacy services. *Legacy* refers to `/etc/rc*.d`, `/etc/init.d`, and `/etc/inittab` scripts, which are popularly used in RHEL environments. Legacy services can continue to work as they did previously, but you will be able to observe these services with SMF. However, to exploit all the advantages of SMF, such as self-healing capabilities or service restart, you need to convert the scripts to SMF manifests.

Service Manifests

Information regarding services, service properties, property groups, and instances is added to the SMF repository. This information is stored in a *service manifest* that SMF uses when managing services as well as when determining the root causes of service failures. The service manifest also describes the conditions under which failed services may be automatically restarted. A separate service manifest is required per service/application. Oracle Solaris provides some service manifests by default. Optionally, you can customize these manifests or write your own for other services.

Leveraging SMF Facilities in Oracle Solaris 11

The Oracle Solaris 11 SMF facility can be leveraged to do many new administrative tasks. For example, many manual configuration tasks can be moved to SMF. This includes the migration of several system, network, and naming service configurations to SMF.

The following key changes are introduced in the latest Oracle Solaris 11 release:

- The `/etc/default/init` file is now read-only on Oracle Solaris.
- Locale and time zone configuration has migrated to SMF. The `svc:/system/timezone:default` SMF service is used to set a system's time zone. Hence, all changes to environment variables should be managed through the new `svc:/system/environment:init` SMF service.

Note: To use the `svc:/system/environment:init` SMF service, make sure that the `skip_init_upgrade` property is set to `true`.

- A system's identify (node name) is now configured through the `config/nodename` service property of the `svc:/system/identity:node` SMF service.

Chapter 9 Infrastructure Differences

Most enterprise applications have a great deal in common—no matter what the application does or who uses it. Almost every enterprise application requires services such as authentication, logging, persistence, security, and so on. In most applications, each service is either developed by the application developer, or reused after customizing the offerings provided by different vendors, or implemented by leveraging the frameworks provided by the operating system.

Applications developed using a framework are interoperable with market standards. Generally, a framework provided by the operating system ensures conformance with the standards, maintainability, and upgradability as well as availability across multiple platforms at lower cost. Software is all about reusability and adapting to change. A framework provides the certainty that you are developing an application that is in full compliance with the business rules, that is structured, and that is portable, maintainable, and upgradable with changing business rules and compliance requirements.

In short, if instead of having a custom-built implementation, your application uses a framework provided by the operating system, for example, a security framework, file system framework, cryptographic framework, or hot-plug framework, migration from one platform to another becomes simple. Most frameworks available on RHEL are available on Oracle Solaris 11 and maintains similar, if not the same, interfaces.

This chapter is intended for developers of system-entry applications that provide authentication, account management, session management, and password management through Pluggable Authentication Modules (PAM) modules. The goal of this chapter is to describe differences in implementation and things to consider while migrating an application from RHEL to Oracle Solaris 11.

Pluggable Authentication Module (PAM)

PAM provides 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`, 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.

RHEL and Oracle Solaris 11 provide similar PAM infrastructures. Though similar functionality is available on both the platforms, there are subtle differences in the two implementations.

PAM Configuration and Differences

The following table lists the PAM-related system administration command differences on the two platforms.

TABLE 9-1. INFRASTRUCTURE DIFFERENCES

RHEL	ORACLE SOLARIS 11
The Linux-PAM configuration file is located in `/etc/pam.conf`. Alternatively, PAM configuration can be set by editing individual configuration files located in the `/etc/pam.d/` directory. The presence of the `/etc/pam.d` directory will cause Linux-PAM to ignore `/etc/pam.conf`.	On Oracle Solaris 11, a configuration file named `/etc/pam.conf` is used to configure the service modules for various system services. On Oracle Solaris, all PAM-related configurations can be done by editing the single file `/etc/pam.conf`. There is no need for the `/etc/pam.d` directory on Oracle Solaris 11.

The supported PAM control values for Linux in the `pam.conf` file and individual PAM configuration files are:
`required`
`requisite`
`sufficient`
`optional`
`include`
`substack`
	On Oracle Solaris, the supported control values in `/etc/pam.conf` file are:
	`required`
	`requisite`
	`sufficient`
	`optional`
	`include`
	`binding`
	
	Note: When binding is specified, if the service module returns `success` and no preceding required modules returned failures, immediately return `success` without calling any subsequent modules.
	If a failure is returned, treat the failure as a required module failure, and continue to process the PAM stack.
	Control value `substack` is not available in Oracle Solaris.

Migrating Custom PAM Modules—Developer's Perspective

To understand the implementation level differences between the two operating systems, let's first look at how PAM works. Table 9-2 lists the sequence of function calls that happen during invocation of a typical PAM module.

TABLE 9-2. SEQUENCE OF FUNCTION CALLS DURING PAM INVOCATION

STEP-BY-STEP SEQUENCE	ACTUAL SEQUENCE OF FUNCTION CALLS

The application initializes the library with a call to `pam_start()`.	The server calls `pam_start`(3) to initialize the PAM library, specify its service name and the target account, and register a suitable conversation function.
	
	`#include <security/pam_appl.h>`
	
	`Int pam_start(const char *service, const char *user,`
	`const struct pam_conv *pam_conv, pam_handle_t **pamh);`

	The server obtains various information relating to the transaction (such as the applicant's user name and the name of the host the client runs on) and submits it to PAM using `pam_set_item`(3).
	
	`#include <security/pam_appl.h>`
	
	`Int pam_set_item(pam_handle_t *pamh, int item_type, const void *item);`

	The server calls `pam_authenticate`(3) to authenticate the applicant.
	
	`#include <security/pam_appl.h>`
	
	`int pam_authenticate(pam_handle_t *pamh, int flags);`

Once a user has been authenticated, the `pam_acct_mgmt()` function is used to establish whether the user is permitted to log in at this time.	The server calls `pam_acct_mgmt`(3) to verify that the requested account is available and valid. If the password is correct but has expired, `pam_acct_mgmt`(3) will return `PAM_NEW_AUTHTOK_REQD` instead of `PAM_SUCCESS`.
	
	`#include <security/pam_appl.h>`
	
	`int pam_acct_mgmt(pam_handle_t *pamh, int flags);`

TABLE 9-2. SEQUENCE OF FUNCTION CALLS DURING PAM INVOCATION

STEP-BY-STEP SEQUENCE	ACTUAL SEQUENCE OF FUNCTION CALLS

Modules of account-management type can be used to restrict users from logging in at certain times of the day or week or for enforcing password expiration. In this case, users are prevented from gaining access to the system until they have successfully updated their password with the `pam_chauthtok()` function.	If the previous step returned `PAM_NEW_AUTHTOK_REQD`, the server now calls `pam_chauthtok`(3) to force the client to change the authentication token for the requested account.
	
	`#include <security/pam_appl.h>`
	
	`Int pam_chauthtok(pam_handle_t *pamh, const int flags);`

`pam_setcred()` establishes and releases the PAM-configurable identity of the user. This can include credentials such as access tickets and supplementary group memberships.	Now that the applicant has been properly authenticated, the server calls `pam_setcred`(3) to establish the credentials of the requested account. It is able to do this because it acts on behalf of the arbitrator and holds the arbitrator's credentials.

`pam_open_session`() and `pam_close_session`() mark the beginning and end of the PAM-authenticated session. Session initialization and termination typically include tasks such as making a system resource available (mounting the user's home directory) and establishing an audit trail.	Once the correct credentials have been established, the server calls `pam_open_session`(3) to set up the session.
	
	`#include <security/pam_appl.h>`
	
	`Int pam_open_session(pam_handle_t *pamh, int flags);`

	Once the server is done serving the client, it calls `pam_close_session`(3) to tear down the session.
	
	`#include <security/pam_appl.h>`
	
	`int pam_close_session(pam_handle_t *pamh, int flags);`

With a call to `pam_end()`, the login application breaks its connection to the PAM library. The PAMs are unloaded, and the dynamically allocated memory is scrubbed and returned to the system.	Finally, the server calls `pam_end`(3) to notify the PAM library that it is done and that it can release whatever resources it has allocated in the course of the transaction.
	
	`#include <security/pam_appl.h>`
	
	`int pam_end(pam_handle_t *pamh, int status);`

Differences in PAM Data Structures and Function Calls

The following table lists the differences in data structures used by PAM on the two platforms.

TABLE 9-3. DATA STRUCTURE AND FUNCTION CALLS

RHEL	ORACLE SOLARIS 11

Inside `pam_start()`, `structure pam_message` is defined as follows:
`struct pam_message {`
`int msg_style;`
`const char *msg;`
`};`
Linux-PAM interprets the `msg` argument as entirely equivalent to the following prototype:
`const struct pam_message *msg[]`
	In Oracle Solaris, `pam_message` is defined as follows:
	`struct pam_message{`
	`int msg_style;`
	`char *msg; // <~~~~~ Not defined as const`
	`};`
	Oracle Solaris PAM implementation interprets this argument as a pointer to a pointer to an array of `num_msg pam_message` structures.

The `pam_set_item()` and `pam_get_item()` functions in Linux are defined under `security/pam_modules.h`.	In Oracle Solaris, `pam_set_item()` and `pam_get_item` are located in `security/pam_appl.h`. `item_type` values supported in `pam_set_item()` are as follows: `PAM_AUSER`, `PAM_AUTHTOK`, `PAM_CONV`, `PAM_OLDAUTHTOK`, `PAM_RESOURCE`, `PAM_RHOST`, `PAM_RUSER`, `PAM_SERVICE`, `PAM_TTY`, `PAM_USER`, `PAM_USER_PROMPT`, and `PAM_REPOSITORY` The `item_type PAM_SERVICE` can be set only by `pam_start()` and is read-only to both applications and service modules. For security reasons, the `item_type PAM_AUTHTOK` and `PAM_OLDAUTHTOK` are available only to the module providers. Return value `PAM_BAD_ITEM` is missing in Oracle Solaris; instead, it has `PAM_OPEN_ERR - dlopen() failed` when dynamically loading a service module. `PAM_SYMBOL_ERR` means a symbol was not found. `PAM_SERVICE_ERR` means there was an error in a service module. `PAM_CONV_ERR` indicates a conversation failure.

Necessary Compiler Option, Linker Options, and Linked Libraries

TABLE 9-4. COMPILATION STEPS AND MAKEFILE CHANGES

RHEL	ORACLE SOLARIS 11
`# gcc -fPIC -c pam_module.c` `# gcc -shared -o pam_module.so pam_module.o –lpam`	`# cc -o pam_module -lpam pam_module.c pam_module_conv.c`

Chapter 10 Packaging and Distributing Applications

With increasing trends towards consolidation using virtualization, system administrators are required to manage large volumes of software often with complex interdependencies on a varied set of hardware platforms. Keeping the systems running smoothly at data centers with complex software dependencies on varied hardware platforms is becoming a complex and challenging task every day. Proper management of system software can help ensure a well-known, tested, and consistent system state across a variety of systems in the data center. For most enterprise IT organizations, significant effort goes into upgrading operating system software to appropriate patch levels to benefit from bug fixes, security updates, or new hardware driver support.

While a simple application might consist of only a few executable files, most enterprise software applications are more complex. A typical application or utility will consist of several executable files, libraries, scripts, configuration files, and documentation notes and guides. All these files—and information about where to place them in the file system and what permissions they have on the installed system—are put in what is referred to as a *package*. There are many packaging formats popularly used by software vendors. Some are easier to use than others. The choice of packaging mechanism depends on the application requirements as well as upgrade and maintainability requirements. Your application packaging needs can be met by simple packaging mechanisms, such as tarballs, or you might need complex and advanced packaging mechanisms, such as RPM or IPS.

Packaging on RHEL

On RHEL, RPM is the powerful package manager that provides the infrastructure for installation, upgrade, and removal of packages. Typically, each package bundles an application along with all the necessary binaries and documentation associated with that application. For example, the Apache Web server comes with a number of configuration files, a large set of documentation files, and the Apache Web server itself. All of this fits into one RPM package. One of the main advantages of the RPM system is that each `.rpm` file holds a complete package. For example, let's look at the GCC package that contains the GNU Compiler Collection version 4.4, which you'll need in order to compile C code.

The file that holds the GCC package is named `gcc-4.4.5-6.el6.rpm`.

Based on the RPM naming convention, this package represents GCC package version 4.4.5, 6th build of an RPM package, for i686 (Intel) architecture systems. The RPM manager is a very powerful tool; with a single command you can copy a `.rpm` file to another Linux system and install it, getting the complete contents of the package, or you can use other commands to remove or update the package.

RPM files mainly contain four sections, as follows:

- **Identification area**:
  The identification area contains file type information, a.k.a. the RPM package, the version of the RPM packing system, and so on.

- **Signature**:
  The signature contains size, checksums, and other such package signatures.
- **Tagged data**:
  Tags are grouped together at the top of the file in a section known as the *preamble*. The tagged data gives information about the contents of the package; this section contains mandatory and optional information (tags), for example, the `NAME` tag (mandatory) holds the package name, the `PRE` tag (optional) holds a pre-installation script (the script that the `rpm` command runs prior to installing the files).

  The tags within a package follow a specific format: `tag:data`. The tag is separated from its associated data by a colon.

  Typical tag types found in an RPM package include the following:

  - Package naming tags
  - Architecture-specific tags
  - Operating system tags
  - Dependency tags
  - Directory-related tags
  - Source and patch tags
  - Descriptive tags

  Under each of these categories, there can be several flags. For example, under the Dependency tag category, there are tags such as the `requires` tag, the `conflicts` tag, the `provides` tag, and son on.
- **Payload**:
  The payload section specifies the files to be installed on the target system.

Categories of RPM Packages

RPM packages can mainly be divided into two categories:

- **Binary packages (RPMs)**:
  A binary RPM is compiled for a particular architecture, for example, the GCC compiler and tools compiled for an Intel i686 target. You would need separate packages for each targeted platform; the same package won't work on other hardware platforms.
- **Source Code Packages (RPMs)**:
  These RPMs packages provide the source code for other packages.

Packaging on Oracle Solaris 11

Oracle Solaris 11 takes a new approach to package management that greatly simplifies the process of managing patches and updates to help reduce the risk of operating system maintenance issues. Based on extensive customer feedback about patch and upgrade processes for Oracle Solaris 10, Oracle engineers completely redesigned the software packaging system in Oracle Solaris 11. Oracle Solaris 11 represents a significant change for system administrators because of its new software packaging model—the Image Packaging System (IPS).

As the Oracle Solaris operating system evolved to include new technologies, such as Oracle Solaris Zones, ZFS, and Service Management Facility, previously used processes for managing system updates and upgrades became more complex. With thousands of operating system instances installed in some of today's large virtualized data centers, manual methods of tracking and installing patches can result in errors that negatively affect application service availability and data center security.

IPS is a comprehensive delivery framework that spans the complete software lifecycle addressing software installation, updates, system upgrades, and the removal of software packages. In contrast to the SVR4 packaging model used in earlier Oracle Solaris releases, IPS eliminates the need for patching. Relying on the use of software repositories, IPS dramatically changes how an administrator updates system and application software. IPS relies on *network accessible* or *locally available* software repositories as a delivery mechanism, which is similar to how other operating systems (for example, RHEL) supply software updates. During a package install, IPS performs automatic dependency checking and adds additional packages, such as libraries, that might be required.

Administrators can easily set up and manage local repositories to deploy both OS and application packages within network-restricted environments. Repositories are also easily mirrored, allowing an administrator to optimize for more efficient access.

The default network-accessible repository for Oracle Solaris 11 is at http://pkg.oracle.com/solaris/release. This repository contains updates for each new release of Oracle Solaris. Significant bug fixes, security updates, and new software might be provided at any time, at Oracle's discretion, for users to install.

The support repository is available at https://pkg.oracle.com/solaris/support. The support repository provides the latest bug fixes and updates. Access is restricted to users with current Oracle Solaris support contracts.

Preparing an Application for Packaging

IPS is a framework that enables you to list, search, install, update, and remove software packages for the Oracle Solaris 11 operating system. A single IPS command can update your image to a new operating system release.

Once your application is built and ready for packaging, the first step in packing is to create a manifest file. The manifest file provides basic metadata about the package, for example, name, version, category, description, what files and directories are included, what dependencies need to be installed for the package, and so on. In the manifest file, packages can also specify what services to restart after the

installation in order to refresh the system configuration. They can also specify actions such as creating users and groups as part of the package installation process.

The package manifest can be divided into three different parts:

- **Package metadata**:
  This section is conceptually similar to "tagged data (header)" in RPM packages. Similar to RPM packages, this will have mandatory and optional elements. Here are the typical set of attributes defined as part of package metada:
  - `pkg.fmri`—package name and version
  - `variant.arch`—the architectures that the package supports
  - `pkg.description`—package description
  - `pkg.summary`—package summary
  - `info.classification`—a grouping scheme used by package manager
- **Package contents**:
  This specifies what files, directories, and links will be installed as part of the package. Here you can also specify things such as the directory on the destination system to which the files will get copied, the user, the group ownership, the permissions, and son on.
- **Package dependencies**:
  Here you can specify the list of files and packages that must be present on the system before installing this package. Typically, dependency generation is composed of two separate steps: First, determine the files on which this software depends. Next, determine the packages that contain the required files. There are IPS commands to generate and validate the dependency information. IPS allows you to specify various types of dependencies on packages:
  - Require—If this type of dependency is specified, the dependent package or its higher versions must be installed before this package can be installed.
  - Optional—If this type of dependency is specified, the dependent package or its higher version is installed if it is available.
  - Conditional—This type of dependency is most often used to install optional extensions to a package if the requisite base packages are present on the system.
  - Group—Group dependency is used to create a group of packages.
  - Origin—This dependency is used to resolve upgrade issues.

Packaging is a vast and intricate subject. In this chapter, we will restrict our scope to major implementation differences between the two platforms to help in the migration from RHEL RPM to Oracle Solaris 11 IPS.

Implementation Differences for Packaging

The following table lists out the basic infrastructure-level differences for packaging on both the platforms.

TABLE 10-1. INFRASTRUCTURE DIFFERENCES

RHEL	ORACLE SOLARIS 11
RPM packages are provided as compressed archive files that contain one or more files, as well as instructions specifying installation information about those files, including the ownerships and permissions that should be applied to each file during installation.	An IPS package is made of a series of actions. Actions are described in the manifest of a package and they are used for defining the files and directories of the package, setting package attributes, declaring dependencies on other packages, creating users and groups, and installing device drivers. Actions represent the installable objects on a system. Every action consists primarily of its name and a key attribute. Together, these refer to a unique object as it follows a version history.

Package files have four-part names, which typically look something like this:
`kernel-smp-2.6.32.9-3.i686.rpm`
Here, the four parts of each name are separated from each other by dashes or periods. The structure of the package file name is as follows:
`name-version-release.architecture.rpm`
	Each IPS package is represented by a Fault Management Resource Identifier (FMRI). For example, the FMRI `pkg://solaris/system/library@0.5.11,5.11-0.151.0.1:20101105T004750Z` consists of the following sequence of information:
	* Scheme: `pkg`
	* Publisher: Oracle Solaris
	* Category: system
	* Package name: `library`
	* Version string, which consists of four components :
	• Component version : `5.11`
	• Build version: `5.11`
	• Branch version: `151.0.1`
	– Timestamp in ISO-8601 basic format: `20101105T004750Z`

Package Administration and Managing Dependencies and Upgrades

Table 10-2 shows the differences in commands and sequence of steps required to administer packages on the two platforms.

TABLE 10-2. SYSTEM ADMINISTRATION

RHEL	ORACLE SOLARIS 11

To install or upgrade a package, use the `-U` command-line option:
`#rpm -U filename.rpm`
The `--test` option tells the `rpm` command to test the installation or upgrade process but not to install the file:
`#rpm -U --test filename.rpm`
	For installing package, use the following:
	`#pkg install`
	To preview an installation without actually doing it, use the following:
	`# pkg install -nv package_name`
	To update and automatically accept license agreements, if there are any, use the following:
	`#pkg update --accept`

To remove a package (*erase* in RPM terminology), use the `–e` command-line option: `#rpm –e package_name`	To remove a package, use the following: `#sudo pkg uninstall package_name`
To list every RPM package installed on your system, use a command like the following. `#rpm –qa`	To list packages, use the following: `#pkg list`
Use the `rpm –q` command to quickly verify a package has been installed: `#rpm –q package_name`	To verify whether a package is installed, use the following: `#pkg verify -v package_name`
The `–i` option with an `rpm query` command tells the `rpm` command to output descriptive information about the package. `#rpm -qi package_name`	To get information about a package, use the following: `#pkg info package_name`
The `–l` (ell) option queries all the files in a package: `#rpm –ql  package_name`	To list files in a package, use the following: `#pkg contents package_name`

Building a Package in Oracle Solaris

To understand the implementation-level differences between the two operating systems, let's look at the sequence of steps you need to follow while creating and publishing a new package. Table 10-3 shows the steps required for creating a package and Table 10-4 shows the steps for publishing a package.

TABLE 10-3. CREATING A PACKAGE

RHEL	ORACLE SOLARIS 11

On RHEL, consider that a `tar` file of the source is created and put in the `/path name/dir-name` directory. The specification (spec) file is created, which contains information about the package and has the following sections:
• The prep section (short for prepare) defines the commands necessary to prepare for the build.
• The build section contains the commands for building the software. Usually, this will include just a few commands, since most of the real instructions appear in the makefile.
• The install section holds the commands necessary to install the newly built application or library.
• The clean section cleans up the files that the commands in the other sections create.
• The files section lists the files that go into the binary RPM along with the defined file attributes.
	Let us look at steps to create a package for a sample application.
	Step 1—In Oracle Solaris 11, first create a workspace and a prototype area where you want to create the package:
	`oracle@solaris_11:~$ cd ~/work`
	`oracle@solaris_11:~/work$ mkdir proto_inst`
	Extract the sample application source code:
	`oracle@solaris_11:~/work$ tar -zxf my_sample_src.tar.gz`
	`oracle@solaris_11:~/work$ ls`
	`my_sample_src/  proto_inst/`
	Step 2—Configure, compile, and install the sample application using the following commands:
	`oracle@solaris_11:~/work$ cd my_sample_src`
	Set the sample application target install drectories:
	`oracle@solaris_11:~/work/my_sample_src $ ./configure exec_prefix=~/work/proto_inst prefix=/usr`
	Compile the sample application sources:
	`oracle@solaris_11X:~/work/my_sample_src $ make`
	Install the sample application:
	`oracle@solaris_11X:~/work/my_sample_src $ make install`
	We are now ready to take the next step to create an IPS package from the files that were just installed in the proto_install area.
	`oracle@solaris_11X:~/work/my_sample_src $ cd ~/work/proto_inst`
	`oracle@solaris_11X:~/work/proto_inst $ls`
	`bin/  doc/  lib/  man/  scripts/`
	`oracle@solaris_11X:~/work/proto_inst $ cd ~/work/`
	`oracle@solaris_11X:~/work $`

	Step 3—Create package metadata:
	`cat > my_sample.txt << EOF`
	`set name=pkg.fmri value=developer/versioning/my_sample@2.4.1,0.1`
	`set name=pkg.description value="my_sample is a sample application"`
	`set name=pkg.summary value="mysample is a popular opensource sample application"`
	`set name=variant.arch value=$(ARCH)`
	`set name=info.classification value="org.opensolaris.category.2008:Development/sample application"`
	`EOF`

TABLE 10-3. CREATING A PACKAGE

RHEL	ORACLE SOLARIS 11
	Step 4—Specify package contents Use the command `pkgsend generate` to parse through proto area recursively and specify what all constitutes the package `oracle@solaris_11X:~/work $ pkgsend generate proto_install | pkgfmt > my_sample.p5m.1`

The `find-requires` and `find-provides` scripts in `/usr/lib/rpm` can determine Perl, Python, and Tcl script dependencies and other dependencies, such as Java package dependencies, automatically. The `find-requires` script determines requires dependencies automatically, and the `find-provides` script determines provides dependencies.	Step 5: Generated package dependencies using the `pkgdepend generate` command:
	`oracle@solaris_11X:~/work $ pkgdepend generate -md proto_install my_sample.p5m.1|pkgfmt > my_sample_depend`
	Step 6: Resolve package dependancies using the following command:
	`oracle@solaris_11:~/work $ pkgdepend resolve my_sample_depend`

The `rpmbuild` command is used to build RPMs:
`$rpmbuild -bBuildStage spec_file`
To build a binary RPM, use the `–bb` option to the `rpmbuild` command:
`$ rpmbuild -bb spec_file`
	After exexuting the above command, my_sample_depend.res will be created. This file can now be used to publish the packge on the IPS repository.

TABLE 10-4. PUBLISHING A PACKAGE

RHEL	ORACLE SOLARIS 11

Once the package is ready, it can be published on a Web server so it can be downloaded by end users like any other download.	Suppose we have a repository named `example.com`.
	To publish the package, ensure that it is read/write, not the default (read-only). To accomplish this, run the following command:
	`oracle@solaris_11:~/work $ svccfg -s pkg/server setprop pkg/readonly=false`
	Restart the server for the changes to take effect:
	`oracle@solaris_11:~/work $ svcadm restart pkg/server:default`

	To publish the package, use the following command: `oracle@solaris_11:~/work $ pkgsend publish -s http://example.com:9001 -d ./proto_inst` *`my_sample_depend`* `PUBLISHED` Now the package has successfully been published.

	We can now check the status of our repository again to determine whether our package got published:
	`oracle@solaris_11:~/work $ pkgrepo info -s http://example.com:9001`
	`PUBLISHER PACKAGES STATUS UPDATED`
	`example.com 1 online date`

Appendix A Security and Privileges

The Oracle Solaris 11 security privileges are defined in Table A-1.

TABLE A-1. ORACLE SOLARIS 11 SECURITY PRIVILEGES

PRIVILEGE NAME	DESCRIPTION
PRIV_CONTRACT_EVENT	Allow a process to request reliable delivery of events to an event endpoint. Allow a process to include events in the critical event `set term` of a template, which could be generated in volume by the user.
PRIV_CONTRACT_OBSERVER	Allow a process to observe contract events generated by contracts created and owned by users other than the process's effective user ID. Allow a process to open contract event endpoints belonging to contracts created and owned by users other than the process's effective user ID.
PRIV_CPC_CPU	Allow a process to access per-CPU hardware performance counters.
PRIV_DTRACE_KERNEL	Allow DTrace kernel-level tracing.
PRIV_DTRACE_PROC	Allow DTrace process-level tracing. Allow process-level tracing probes to be placed and enabled in processes to which the user has permissions.
PRIV_DTRACE_USER	Allow DTrace user-level tracing. Allow use of the `syscall` and `profile` DTrace providers to examine processes to which the user has permissions.
PRIV_FILE_CHOWN	Allow a process to change a file's owner user ID. Allow a process to change a file's group ID to one other than the process's effective group ID or one of the process's supplemental group IDs.
PRIV_FILE_CHOWN_SELF	Allow a process to give away its files. A process with this privilege will run as if {_POSIX_CHOWN_RESTRICTED} is not in effect.
PRIV_FILE_DAC_EXECUTE	Allow a process to execute an executable file whose permission bits or ACL would otherwise disallow the process execute permission.
PRIV_FILE_DAC_READ	Allow a process to read a file or directory whose permission bits or ACL would otherwise disallow the process read permission.

TABLE A-1. ORACLE SOLARIS 11 SECURITY PRIVILEGES

PRIVILEGE NAME	DESCRIPTION
PRIV_FILE_DAC_SEARCH	Allow a process to search a directory whose permission bits or ACL would not otherwise allow the process search permission.
PRIV_FILE_DAC_WRITE	Allow a process to write a file or directory whose permission bits or ACL do not allow the process write permission. All privileges are required to write files owned by UID 0 in the absence of an effective UID of 0.
PRIV_FILE_DOWNGRADE_SL	Allow a process to set the sensitivity label of a file or directory to a sensitivity label that does not dominate the existing sensitivity label. This privilege is interpreted only if the system is configured with Trusted Extensions.
PRIV_FILE_LINK_ANY	Allow a process to create hardlinks to files owned by a UID different from the process's effective UID.
PRIV_FILE_OWNER	Allow a process that is not the owner of a file to modify that file's access and modification times. Allow a process that is not the owner of a directory to modify that directory's access and modification times. Allow a process that is not the owner of a file or directory to remove or rename a file or directory whose parent directory has the “save text image after execution” (sticky) bit set. Allow a process that is not the owner of a file to mount a `namefs` upon that file. Allow a process that is not the owner of a file or directory to modify that file's or directory's permission bits or ACL.
PRIV_FILE_SETID	Allow a process to change the ownership of a file or write to a file without the `set-user-ID` and `set-group-ID` bits being cleared. Allow a process to set the `set-group-ID` bit on a file or directory whose group is not the process's effective group or one of the process's supplemental groups. Allow a process to set the `set-user-ID` bit on a file with different ownership in the presence of `PRIV_FILE_OWNER`. Additional restrictions apply when creating or modifying a setuid 0 file.
PRIV_FILE_UPGRADE_SL	Allow a process to set the sensitivity label of a file or directory to a sensitivity label that dominates the existing sensitivity label.
PRIV_GRAPHICS_ACCESS	Allow a process to make privileged ioctls to graphics devices. Typically only an xserver process needs to have this privilege. A process with this privilege is also allowed to perform privileged graphics device mappings.
PRIV_GRAPHICS_MAP	Allow a process to perform privileged mappings through a graphics device.

TABLE A-1. ORACLE SOLARIS 11 SECURITY PRIVILEGES

PRIVILEGE NAME	DESCRIPTION
PRIV_IPC_DAC_READ	Allow a process to read a System V IPC message queue, semaphore set, or shared memory segment whose permission bits would not otherwise allow the process read permission.
PRIV_IPC_DAC_WRITE	Allow a process to write a System V IPC message queue, semaphore set, or shared memory segment whose permission bits would not otherwise allow the process write permission.
PRIV_IPC_OWNER	Allow a process that is not the owner of a System V IPC message queue, semaphore set, or shared memory segment to remove, change ownership of, or change the permission bits of the message queue, semaphore set, or shared memory segment.
PRIV_NET_ACCESS	Allow a process to open a TCP, UDP, SDP, or SCTP network endpoint.
PRIV_NET_BINDMLP	Allow a process to bind to a port that is configured as a multilevel port (MLP) for the process's zone. This privilege applies to both shared address and zone-specific address MLPs. See `tnzonecfg`(4) from the Trusted Extensions manual pages for information on configuring MLP ports. This privilege is interpreted only if the system is configured with Trusted Extensions.
PRIV_NET_ICMPACCESS	Allow a process to send and receive ICMP packets.
PRIV_NET_MAC_AWARE	Allow a process to set the NET_MAC_AWARE process flag by using `setpflags(2)`. This privilege also allows a process to set the SO_MAC_EXEMPT socket option by using `setsockopt(3SOCKET)`. The NET_MAC_AWARE process flag and the SO_MAC_EXEMPT socket option both allow a local process to communicate with an unlabeled peer if the local process's label dominates the peer's default label, or if the local process runs in the global zone. This privilege is interpreted only if the system is configured with Trusted Extensions.
PRIV_NET_PRIVADDR	Allow a process to bind to a privileged port number. The privilege port numbers are 1–1023 (the traditional UNIX privileged ports) as well as those ports marked as `udp/tcp_extra_priv_ports` with the exception of the ports reserved for use by NFS.
PRIV_NET_RAWACCESS	Allow a process to have direct access to the network layer.
PRIV_PROC_AUDIT	Allow a process to generate audit records. Allow a process to get its own audit pre-selection information.

TABLE A-1. ORACLE SOLARIS 11 SECURITY PRIVILEGES

PRIVILEGE NAME	DESCRIPTION
PRIV_PROC_CHROOT	Allow a process to change its root directory.
PRIV_PROC_CLOCK_HIGHRES	Allow a process to use high-resolution timers.
PRIV_PROC_EXEC	Allow a process to call exec(2).
PRIV_PROC_FORK	Allow a process to call fork(2), fork1(2), or vfork(2).
PRIV_PROC_INFO	Allow a process to examine the status of processes other than those to which it can send signals. Processes that cannot be examined cannot be seen in /proc and appear not to exist.
PRIV_PROC_LOCK_MEMORY	Allow a process to lock pages in physical memory.
PRIV_PROC_OWNER	Allow a process to send signals to other processes and inspect and modify the process state in other processes, regardless of ownership. When modifying another process, additional restrictions apply: The effective privilege set of the attaching process must be a superset of the target process's effective, permitted, and inheritable sets; the limit set must be a superset of the target's limit set; if the target process has any UID set to 0, all privileges must be asserted unless the effective UID is 0. Allow a process to bind arbitrary processes to CPUs.
PRIV_PROC_PRIOCNTL	Allow a process to elevate its priority above its current level. Allow a process to change its scheduling class to any scheduling class, including the RT class.
PRIV_PROC_SESSION	Allow a process to send signals or trace processes outside its session.
PRIV_PROC_SETID	Allow a process to set its UIDs at will, assuming UID 0 requires all privileges to be asserted.
PRIV_PROC_TASKID	Allow a process to assign a new task ID to the calling process.
PRIV_PROC_ZONE	Allow a process to trace or send signals to processes in other zones. See zones(5).
PRIV_SYS_ACCT	Allow a process to enable and disable and manage accounting through acct(2).
PRIV_SYS_ADMIN	Allow a process to perform system administration tasks such as setting the node and domain name and specifying coreadm(1M) and nscd(1M) settings.

TABLE A-1. ORACLE SOLARIS 11 SECURITY PRIVILEGES

PRIVILEGE NAME	DESCRIPTION
PRIV_SYS_AUDIT	Allow a process to start the (kernel) audit daemon. Allow a process to view and set audit state (audit user ID, audit terminal ID, audit sessions ID, audit pre-selection mask). Allow a process to turn off and on auditing. Allow a process to configure the audit parameters (cache and queue sizes, event-to-class mappings, and policy options).
PRIV_SYS_CONFIG	Allow a process to perform various system configuration tasks. Allow file system–specific administrative procedures, such as file system configuration ioctls, quota calls, creation and deletion of snapshots, and manipulating the PCFS boot sector.
PRIV_SYS_DEVICES	Allow a process to create device-special files. Allow a process to successfully call a kernel module that calls the kernel drv_priv(9F) function to check for allowed access. Allow a process to open the real console device directly. Allow a process to open devices that have been exclusively opened.
PRIV_SYS_IPC_CONFIG	Allow a process to increase the size of a System V IPC message queue buffer.
PRIV_SYS_LINKDIR	Allow a process to unlink and link directories.
PRIV_SYS_MOUNT	Allow a process to mount and unmount file systems that would otherwise be restricted (that is, most file systems except namefs). Allow a process to add and remove swap devices.
PRIV_SYS_IP_CONFIG	Allow a process to configure a system's network interfaces and routes. Allow a process to configure network parameters for TCP/IP using ndd. Allow a process access to otherwise restricted TCP/IP information using ndd. Allow a process to configure IPsec. Allows a process to pop anchored STREAMs modules with a matching zoneid.
PRIV_SYS_NET_CONFIG	Allow a process to do all that PRIV_SYS_IP_CONFIG allows, plus the following: Push the rpcmod STREAMS module, insert and remove STREAMS modules on locations other than the top of the module stack, and configure data links (NICs).
PRIV_SYS_NFS	Allow a process to provide NFS service: Start NFS kernel threads, perform NFS-locking operations, bind to NFS reserved ports 2049 (nfs) and port 4045 (lockd).

TABLE A-1. ORACLE SOLARIS 11 SECURITY PRIVILEGES

PRIVILEGE NAME	DESCRIPTION
PRIV_SYS_RES_CONFIG	Allow a process to create and delete processor sets, assign CPUs to processor sets and override the PSET_NOESCAPE property. Allow a process to change the operational status of CPUs in the system using p_online(2). Allow a process to configure file system quotas. Allow a process to configure resource pools and bind processes to pools.
PRIV_SYS_RESOURCE	Allow a process to exceed the resource limits imposed on it by setrlimit(2) and setrctl(2).
PRIV_SYS_SUSER_COMPAT	Allow a process to successfully call a third-party loadable module that calls the kernel suser() function to check for allowed access. This privilege exists only for third-party loadable module compatibility and is not used by Oracle Solaris proper.
PRIV_SYS_TIME	Allow a process to manipulate system time using any of the appropriate system calls: stime(2), adjtime(2), and ntp_adjtime(2).
PRIV_SYS_TRANS_LABEL	Allow a process to translate labels that are not dominated by the process's sensitivity label to and from an external string form. This privilege is interpreted only if the system is configured with Trusted Extensions.
PRIV_WIN_COLORMAP	Allow a process to override colormap restrictions. Allow a process to install or remove colormaps. Allow a process to retrieve colormap cell entries allocated by other processes. This privilege is interpreted only if the system is configured with Trusted Extensions.
PRIV_WIN_CONFIG	Allow a process to configure or destroy resources that are permanently retained by the X server. Allow a process to use SetScreenSaver to set the screen saver timeout value. Allow a process to use ChangeHosts to modify the display access control list. Allow a process to use GrabServer. Allow a process to use the SetCloseDownMode request that can retain window, pixmap, colormap, property, cursor, font, or graphic context resources. This privilege is interpreted only if the system is configured with Trusted Extensions.

TABLE A-1. ORACLE SOLARIS 11 SECURITY PRIVILEGES

PRIVILEGE NAME	DESCRIPTION
PRIV_WIN_DAC_READ	Allow a process to read from a window resource that it does not own (one that has a different user ID). This privilege is interpreted only if the system is configured with Trusted Extensions.
PRIV_WIN_DAC_WRITE	Allow a process to write to or create a window resource that it does not own (one that has a different user ID). A newly created window property is created with the window's user ID. This privilege is interpreted only if the system is configured with Trusted Extensions.
PRIV_WIN_DEVICES	Allow a process to perform operations on window input devices. Allow a process to get and set keyboard and pointer controls. Allow a process to modify pointer button and key mappings. This privilege is interpreted only if the system is configured with Trusted Extensions.
PRIV_WIN_DGA	Allow a process to use the direct graphics access (DGA) X protocol extensions. Direct process access to the frame buffer is still required. Thus, the process must have MAC and DAC privileges that allow access to the frame buffer, or the frame buffer must be allocated to the process. This privilege is interpreted only if the system is configured with Trusted Extensions.
PRIV_WIN_DOWNGRADE_SL	Allow a process to set the sensitivity label of a window resource to a sensitivity label that does not dominate the existing sensitivity label. This privilege is interpreted only if the system is configured with Trusted Extensions.
PRIV_WIN_FONTPATH	Allow a process to set a font path. This privilege is interpreted only if the system is configured with Trusted Extensions.
PRIV_WIN_MAC_READ	Allow a process to read from a window resource whose sensitivity label is not equal to the process sensitivity label. This privilege is interpreted only if the system is configured with Trusted Extensions.

TABLE A-1. ORACLE SOLARIS 11 SECURITY PRIVILEGES

PRIVILEGE NAME	DESCRIPTION
PRIV_WIN_MAC_WRITE	Allow a process to create a window resource whose sensitivity label is not equal to the process sensitivity label. A newly created window property is created with the window's sensitivity label. This privilege is interpreted only if the system is configured with Trusted Extensions.
PRIV_WIN_SELECTION	Allow a process to request inter-window data moves without the intervention of the selection confirmer. This privilege is interpreted only if the system is configured with Trusted Extensions.
PRIV_WIN_UPGRADE_SL	Allow a process to set the sensitivity label of a window resource to a sensitivity label that dominates the existing sensitivity label. This privilege is interpreted only if the system is configured with Trusted Extensions.

Appendix B GCC to Oracle Solaris Studio Compiler Flag Mapping

The compiler flags shown in Table B-1 remain the same for both GCC and Oracle Solaris Studio.

TABLE B-1. COMPILER FLAGS THAT REMAIN THE SAME

GCC OPTION	ORACLE SOLARIS STUDIO 12.3 OPTION	DESCRIPTION
`-###`	`-###`	Similar to `-#`, but the stages are not actually executed.
`-A name=token`	`-Aname[ (token)]`	Associate *name* as a predicate with the specified *token* as if by the `#assert` preprocessing directive.
`-C`	`-C`	Prevents the preprocessor from removing comments.
`-c`	`-c`	Directs the compiler to compile, but not to link.
`-Dname[ =val]`	`-Dname[ =val]`	Defines a preprocessor macro.
`-E`	`-E`	Runs only the preprocessor on source files.
`-Wc, arg`	`-Wc, arg`	Tells the compiler to pass *arg* as an argument to the tool named by *c*.
`-w`	`-w`	Suppresses warnings.
`-S`	`-S`	Directs the compiler to produce an assembly source file.
`-s`	`-s`	Removes all symbolic debugging information from the output file. For Intel, use the Linux strip utility on the executable.
`-Uname`	`-Uname`	Undefines the preprocessor symbol *name*.
`-lname`	`-lname`	Links with `library libname.a` (or `.so`).
`-O`	`-O`	Turns on the default optimization level (`-x02` for Oracle Solaris, `-01` for GCC, `-02` for Intel).
`-o file`	`-o file`	Specifies the output file.
`-p`	`-p`	Prepares object code to collect data for profiling with `prof` (1).
`-Ldir`	`-Ldir`	Adds *dir* to the library search directory.
`-g`	`-g`	Generates debugging information.
`-H`	`-H`	Prints the path name of each of the header files being compiled.
`-I[ dir]`	`-I[ dir]`	Adds an include search directory.
`-I-`	`-I-`	Any directories you specify with `–I` options before the `-I-` option are searched only for the case of `#include "file"`; they are not searched for `#include <file>`.

TABLE B-1. COMPILER FLAGS THAT REMAIN THE SAME

GCC OPTION	ORACLE SOLARIS STUDIO 12.3 OPTION	DESCRIPTION
`-O[ n]`	`-xO[ n]`	Selects the optimization level.

Red Hat Enterprise Linux to Oracle Solaris Porting Guide
July 2012, 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