The software described in this documentation is either in Extended Support or Sustaining Support. See
https://www.oracle.com/us/support/library/enterprise-linux-support-policies-069172.pdf for more information.
Oracle recommends that you upgrade the software described by this documentation as soon as possible.

Oracle® Linux 6

Porting Guide

ORACLE

E52461-08
March 2021

The software described in this documentation is either in Extended Support or Sustaining Support. See https://www.oracle.com/us/support/library/enterprise-linux-support-policies-069172.pdf for more information. Oracle recommends that you upgrade the software described by this documentation as soon as possible.

Oracle Legal Notices

Copyright © 2014, 2021, Oracle and/or its affiliates.

This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish, or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited.

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

If this is software or related documentation that is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, then the following notice is applicable:

U.S. GOVERNMENT END USERS: Oracle programs (including any operating system, integrated software, any programs embedded, installed or activated on delivered hardware, and modifications of such programs) and Oracle computer documentation or other Oracle data delivered to or accessed by U.S. Government end users are "commercial computer software" or "commercial computer software documentation" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, the use, reproduction, duplication, release, display, disclosure, modification, preparation of derivative works, and/or adaptation of i) Oracle programs (including any operating system, integrated software, any programs embedded, installed or activated on delivered hardware, and modifications of such programs), ii) Oracle computer documentation and/or iii) other Oracle data, is subject to the rights and limitations specified in the license contained in the applicable contract. The terms governing the U.S. Government's use of Oracle cloud services are defined by the applicable contract for such services. No other rights are granted to the U.S. Government.

This software or hardware is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications that may create a risk of personal injury. If you use this software or hardware in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy, and other measures to ensure its safe use. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software or hardware in dangerous applications.

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Intel and Intel Inside are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. AMD, Epyc, and the AMD logo are trademarks or registered trademarks of Advanced Micro Devices. UNIX is a registered trademark of The Open Group.

This software or hardware and documentation may provide access to or information about content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services unless otherwise set forth in an applicable agreement between you and Oracle. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services, except as set forth in an applicable agreement between you and Oracle.

Abstract

This guide describes how to resolve common issues that arise when migrating applications to Oracle Linux. It describes potential similarities and differences in architecture, system calls, tools, utilities, development environments,

The software described in this documentation is either in Extended Support or Sustaining Support. See https://www.oracle.com/us/support/library/enterprise-linux-support-policies-069172.pdf for more information.
Oracle recommends that you upgrade the software described by this documentation as soon as possible.

and operating system features. Wherever possible, solutions and workarounds are suggested for addressing porting issues that commonly arise.

Document generated on: 2021-03-31 (revision: 11688)

The software described in this documentation is either in Extended Support or Sustaining Support. See https://www.oracle.com/us/support/library/enterprise-linux-support-policies-069172.pdf for more information. Oracle recommends that you upgrade the software described by this documentation as soon as possible.

Table of Contents

Preface vii
1 Introduction 1
1.1 Similarities and Differences Between Oracle Linux and UNIX Operating Systems 1
1.2 Tools and Development Environment 1
1.3 GNU Utilities, Popular Developer Tools, and Open Source Software 1
1.4 Advantages of Porting to Oracle Linux 2
2 The Porting Process 3
2.1 Migration Steps 3
2.2 Recommended Strategy 4
2.3 Assessing the Application Porting Effort 4
2.3.1 Limiting the Scope 4
2.3.2 Classifying Code 5
2.3.3 Scripts and Other Portable Components 5
2.3.4 Build Environment Dependencies 5
2.3.5 Assess Food Chain Dependencies 6
2.4 Data Migration Considerations 6
2.4.1 Data Portability, Well-Known Issues, and Solutions 7
2.5 Verifying Applications 7
2.5.1 Using Gcov to Analyze Code Coverage 8
2.5.2 Using Valgrind to Detect Memory Access Errors and Leaks 8
3 Operating System Considerations 9
3.1 Storage Order and Alignment 9
3.2 Data Structures and Sizes 10
3.3 Compiler Options and Portability of Code 11
3.4 Byte Ordering 11
3.5 Data Conversion for Interoperability 12
3.5.1 Low-Level Code, Bit-Level Operations 12
3.6 System Call Mapping 12
4 Application Development Environment 23
4.1 GNU Compiler Collection 23
4.2 Oracle Solaris Studio for Oracle Linux 23
4.3 Optimizing gcc Compilation 24
4.4 Open Source Software Libraries 25
4.5 Debugging Applications 25
4.6 Identifying Issues Using DTrace 26
5 Threads and Multiprocessing 27
5.1 POSIX Compliance 27
5.2 Threading Model 27
5.3 Differences Between Implementations of Pthreads 27
5.4 Thread Attributes 28
5.5 Signals in Threaded Applications 28
5.6 OpenMP Support 29
5.7 Auto Parallelization and Compile-Time Optimizations 29
5.8 Using the Thread Analyzer 29
6 Migrating Device Drivers 31
6.1 Considerations for Porting Device Drivers 31
6.2 Reading and Writing Data from or to User Space 32
6.3 About Handling Access to Shared Resources 32
6.4 About the Bus Model 33
6.5 About Character Device Drivers 35

The software described in this documentation is either in Extended Support or Sustaining Support. See https://www.oracle.com/us/support/library/enterprise-linux-support-policies-069172.pdf for more information.
Oracle recommends that you upgrade the software described by this documentation as soon as possible.

6.6 About Block Device Drivers 37
6.7 About Network Device Drivers 38
6.8 About USB Device Drivers 41
6.9 About the Sysfs File System 42
6.10 Loading Device Drivers as Kernel Modules 42
7 Security 43
7.1 Physical Security 43
7.2 Delegate Minimal Privileges as Appropriate 43
7.3 About Discretionary and Mandatory Access Control Policies 44
7.4 About Targeted and Multilevel Security Policies 44
7.5 About Security Contexts and Users 45
7.6 Ensure Strong Defenses 45
7.7 Encryption Algorithms, Mechanisms, and Mapping 46
8 Runtime Environment 47
8.1 Runtime Limits 47
8.2 Migrating Scripts 47
8.3 Managing Services 47
9 Pluggable Authentication Modules 49
9.1 About Pluggable Authentication Module (PAM) 49
9.2 About PAM Operation for an Application 50
9.3 PAM Implementation Differences 51
10 Packaging and Distributing Applications 53
10.1 About RPM Packaging 53
10.1.1 About RPM Categories 54
10.1.2 Administering RPMs 54
10.1.3 Building an RPM 55
10.2 About Oracle Enterprise Manager 56
10.3 About Spacewalk 57

The software described in this documentation is either in Extended Support or Sustaining Support. See https://www.oracle.com/us/support/library/enterprise-linux-support-policies-069172.pdf for more information. Oracle recommends that you upgrade the software described by this documentation as soon as possible.

Preface

Oracle® Linux 6: Porting Guide describes how to resolve common issues that arise when migrating applications to Oracle Linux. It describes potential similarities and differences in architecture, system calls, tools, utilities, development environments, and operating system features. Wherever possible, solutions and workarounds are suggested for addressing porting issues that commonly arise.

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 guide also includes best practices to help developers get the most out of their applications when running them on Oracle Linux. 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, this guide avoids going too deep into the specifics of a given problem. Instead, pointers are provided to additional relevant information for further reading. Oracle strongly advises both novice users and those familiar with the Oracle Linux operating system to use manual pages to obtain accurate and detailed information about Oracle Linux and its features.

Audience

This guide is primarily aimed at experienced application developers and device driver writers. It is expected that the reader should have an in-depth knowledge of programming in a language such as C or C++ and the programming environments that are typically used for developing modern software applications. Device driver writers are expected to have an in-depth knowledge of the internals of the operating systems for which they want to develop drivers.

Document Organization

The document is organized as follows:

- Chapter 1, *Introduction* provides an overview of porting software to Oracle Linux.
- Chapter 2, *The Porting Process* provides an overview of the steps that are typically required when porting software to Oracle Linux.
- Chapter 3, *Operating System Considerations* discusses the various aspects that must be considered during migration between platform architectures.
- Chapter 4, *Application Development Environment* provides more information about the GNU Compiler Collection (GCC) and Oracle Solaris Studio 12.x development environments.
- Chapter 5, *Threads and Multiprocessing* provides information about factors that you should take into consideration when porting threaded applications in a multiprocessor environment.
- Chapter 6, *Migrating Device Drivers* provides an overview of how Oracle Linux device drivers are usually implemented and contrasts this approach with that usually encountered on UNIX-like operating systems.
- Chapter 7, *Security* discusses some aspects of ensuring system security on Oracle Linux.
- Chapter 8, *Runtime Environment* discusses some differences that you might encounter in the runtime environment on Oracle Linux.

The software described in this documentation is either in Extended Support or Sustaining Support. See https://www.oracle.com/us/support/library/enterprise-linux-support-policies-069172.pdf for more information.
Oracle recommends that you upgrade the software described by this documentation as soon as possible.

- Chapter 9, *Pluggable Authentication Modules* describes differences in implementation and points that you should consider when migrating an application that uses Pluggable Authentication Modules (PAM) to Oracle Linux.
- Chapter 10, *Packaging and Distributing Applications* provides an overview of how to administer and create RPMs on Oracle Linux. It also provides information about the Oracle Enterprise Manager and Spacewalk IT management products.

Related Documents

The documentation for this product is available at:

Oracle® Linux Documentation

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
`monospace`	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at
https://www.oracle.com/corporate/accessibility/.

Access to Oracle Support for Accessibility

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit
https://www.oracle.com/corporate/accessibility/learning-support.html#support-tab.

Diversity and Inclusion

Oracle is fully committed to diversity and inclusion. Oracle respects and values having a diverse workforce that increases thought leadership and innovation. As part of our initiative to build a more inclusive culture that positively impacts our employees, customers, and partners, we are working to remove insensitive terms from our products and documentation. We are also mindful of the necessity to maintain compatibility with our customers' existing technologies and the need to ensure continuity of service as Oracle's offerings and industry standards evolve. Because of these technical constraints, our effort to remove insensitive terms is ongoing and will take time and external cooperation.

The software described in this documentation is either in Extended Support or Sustaining Support. See https://www.oracle.com/us/support/library/enterprise-linux-support-policies-069172.pdf for more information. Oracle recommends that you upgrade the software described by this documentation as soon as possible.

Chapter 1 Introduction

Table of Contents

1.1 Similarities and Differences Between Oracle Linux and UNIX Operating Systems 1
1.2 Tools and Development Environment 1
1.3 GNU Utilities, Popular Developer Tools, and Open Source Software 1
1.4 Advantages of Porting to Oracle Linux 2

This chapter provides an overview of porting software to Oracle Linux.

1.1 Similarities and Differences Between Oracle Linux and UNIX Operating Systems

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.

Oracle Linux 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.

Provided that the source operating system is POSIX-compliant, there should be little difficulty porting code to Oracle Linux, as long as specific operating extensions are not used. If such extensions are used, an equivalent Oracle Linux extension will need to be used or the extensions will need to be manually ported in order to achieve the desired result.

1.2 Tools and Development Environment

Oracle Linux is a standards-based UNIX operating system that provides a development environment very similar to many modern UNIX operating systems. Most of the popular development and scripting tools used by Linux developers are also available on such operating systems. For example, the default shell for the `root` user on Oracle Linux is `bash` (version 4.1.x) and the Korn shell is available on Oracle Linux. The availability of the same shells on both platforms makes it easier to migrate scripts to Oracle Linux.

Oracle Linux ships with hundreds of standard commands, tools, utilities, and services. These packages are built from the same open source code base that feeds mainstream Linux. Oracle Linux and other UNIX-like operating systems might be similar in many ways. For example, some operating systems offer support for GNU tools and use X Windows as the GUI for their desktop interface. If the shell and development tools are similar on the source platform, it is very easy to move to Oracle Linux as an end user or a developer.

1.3 GNU Utilities, Popular Developer Tools, and Open Source Software

Given that Oracle Linux provides a significant baseline in terms of commands, tools, libraries, platform services, and software development environment, the task of porting an application to Oracle Linux 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

The software described in this documentation is either in Extended Support or Sustaining Support. See https://www.oracle.com/us/support/library/enterprise-linux-support-policies-069172.pdf for more information.
Oracle recommends that you upgrade the software described by this documentation as soon as possible.

has made Oracle Solaris Studio 12.x compilers and tools available on Oracle Solaris, Oracle Linux, and Red Hat Enterprise Linux platforms. The GNU Compiler Collection (GCC) and other GNU tools are also available on these platforms.

Most popular open source software is either already available on Oracle Linux, or the ported binaries as well as source code are available from various popular open source repositories.

1.4 Advantages of Porting to Oracle Linux

There are many advantages moving to Oracle Linux. Oracle Linux is part of Oracle's integrated software stack in which components are optimized to work together to enhance performance, availability, and security.

The following key capabilities of Oracle Linux can make a big difference, particularly in demanding environment for large enterprise-wide deployments.

- Complete integrated software stack optimized to work together.
- 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 (research and development expenditure).
- Oracle's commitment for long-term investments.
- Comprehensive 24x7 support from Oracle.
- DTrace, the dynamic tracing facility, allows you to explore your system to understand how it works, to track down performance problems across many layers of software, or to locate the causes of aberrant behavior.
- Oracle Ksplice updates your systems with the latest kernel security errata and other critical updates without requiring a reboot.
- The Unbreakable Enterprise Kernel (UEK) is a modern, high performance kernel that provides a wide range of features and improvements that support enterprise workloads.

The software described in this documentation is either in Extended Support or Sustaining Support. See https://www.oracle.com/us/support/library/enterprise-linux-support-policies-069172.pdf for more information.
Oracle recommends that you upgrade the software described by this documentation as soon as possible.

Chapter 2 The Porting Process

Table of Contents

2.1 Migration Steps 3
2.2 Recommended Strategy 4
2.3 Assessing the Application Porting Effort 4
    2.3.1 Limiting the Scope 4
    2.3.2 Classifying Code 5
    2.3.3 Scripts and Other Portable Components 5
    2.3.4 Build Environment Dependencies 5
    2.3.5 Assess Food Chain Dependencies 6
2.4 Data Migration Considerations 6
    2.4.1 Data Portability, Well-Known Issues, and Solutions 7
2.5 Verifying Applications 7
    2.5.1 Using Gcov to Analyze Code Coverage 8
    2.5.2 Using Valgrind to Detect Memory Access Errors and Leaks 8

This chapter provides an overview of the steps that are typically required when porting software to Oracle Linux.

2.1 Migration Steps

If both the source and the target operating systems conform to standards and suitable tools are available to aid migration, the migration process is usually straightforward and problem-free. However, it is important to plan the migration systematically, so as to isolate the areas where more effort might be needed.

The following steps are typical of most migration projects:

- Planning phase (making the right choices to minimize porting efforts):
  - Assess application porting.
  - Assess the current environment.
  - Estimate the migration effort.
  - Choose the right tools and build infrastructure.
  - Incorporate Oracle-recommended tools and following best practices.
- Execution phase (porting):
  - Module-wise porting.
  - Unit testing.
  - Integration testing.
- Validation, testing, and certification phase:
  - Functional testing.

The software described in this documentation is either in Extended Support or Sustaining Support. See https://www.oracle.com/us/support/library/enterprise-linux-support-policies-069172.pdf for more information.
Oracle recommends that you upgrade the software described by this documentation as soon as possible.

- System testing.
- Stress tests, soak tests, and long-haul test.

- Deployment phase (getting the most from the new platform):
  - Deployment of applications on the new architecture.
  - Performance tuning.

2.2 Recommended Strategy

Oracle Linux ships with hundreds of popular Linux utilities and tools. Many popular GNU utilities, libraries, and applications are available as optional installable packages and can be installed based on specific application requirements. For example, scripting languages such as gnu-coreutils, gnu-findutils, binutils, glib2, gtk2, Perl, Bison, Ruby, Python, and PHP, as well as Tcl/Tk libraries, GNU Emacs, Apache HTTP server, the GCC compiler, and many other development tools are available.

For the migration phases, you should estimate the time required for completing the tasks. The amount of effort required does not usually depend on the target platform. For example, the times required for functional, soak, and stress testing are usually fairly consistent across platforms. The area that potentially poses the maximum uncertainly is the complexities that are involved in porting the code. This uncertainty can be minimized by spending sufficient time and effort during the planning phase.

Although the source and target operating systems might be similar in many aspects, you might observe subtle differences when it comes to porting. Hence, the effort required for the transition to Oracle Linux can vary greatly due to the composition of application components as well as the programming language and tools that the various application subcomponents use. To arrive at an estimate for the porting effort, it is important to classify the various application subcomponents based on their implementation complexities.

2.3 Assessing 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 Linux. For custom, quickly evolving applications, it is important to freeze a snapshot of the application source code and associated infrastructure that can 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.

2.3.1 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

The software described in this documentation is either in Extended Support or Sustaining Support. See https://www.oracle.com/us/support/library/enterprise-linux-support-policies-069172.pdf for more information.
Oracle recommends that you upgrade the software described by this documentation as soon as possible.

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.

2.3.2 Classifying Code

Before starting the actual porting process, 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.

As a rule of thumb, code written in Java, Perl scripts, and shell scripts should present fewer challenges compared to native modules written in the C, C++, or Visual C programming languages. However, 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.

2.3.3 Scripts and Other Portable Components

The Perl, PHP, and Python utilities are popular as scripting tools because of their power and flexibility as well as their 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 scripts, check the program that executes the script for the following conditions:

- Whether the program is available on Oracle Linux.
- Whether the program is in a different location and the location is not in the user's path.
- Whether multiple implementations of the program are available on the system and `PATH` is picking up the right one.
- Whether the program uses an option that does not exist on Oracle Linux.
- Whether the program uses an option that has different functionality on Oracle Linux.
- Whether the output of the program is different and is redirected.

2.3.4 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 all popular open-source build tools (GNU/GPL) and utilities are available on Oracle Linux. It is much easier to transition to Oracle Linux 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).

The software described in this documentation is either in Extended Support or Sustaining Support. See https://www.oracle.com/us/support/library/enterprise-linux-support-policies-069172.pdf for more information.
Oracle recommends that you upgrade the software described by this documentation as soon as possible.

- Tools used by the applications.
- Command-line options provided by the tools.

2.3.5 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 these dependencies on the Oracle Linux platform, because sometimes the availability of a third-party dependency can become a limiting factor. Below are some guidelines that should not only help to reduce migration effort but also help the applications work better on Oracle Linux:

- 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 Linux 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 Linux features, infrastructure, and tools that can provide similar functionality.
- Look for alternatives from different venders providing similar functionality.

2.4 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, and constraints.
- Migration of associated infrastructure, which includes migration of stored procedures, database triggers, SQL queries, and functions.

If the migrated data is to be readable on the target system, data conversion from one format to another, is an important component of any porting effort. 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, most systems, including Oracle Linux systems, 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 such services, you can realize significant time and cost savings during the migration and testing process.

The software described in this documentation is either in Extended Support or Sustaining Support. See https://www.oracle.com/us/support/library/enterprise-linux-support-policies-069172.pdf for more information.
Oracle recommends that you upgrade the software described by this documentation as soon as possible.

Oracle Linux provides 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. If you are already familiar with another Linux environment, you can usually work seamlessly on Oracle Linux without having to move from your 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.

2.4.1 Data Portability, Well-Known Issues, and Solutions

File systems are neutral to endianness in general, and swapping files is not an issue between SPARC/RISC and x86/x86-64 versions of the same operating system. However, applications storing raw data that needs to be shared across platforms can become an issue.

For example, if an application on a SPARC platform 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 a system that is based on an x86-64 processor can create problems regarding the endianness of the data. Binary (raw) data stored in a file is generally not transferable between SPARC/RISC and x86/x86-64 platforms.

Applications that share data 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 required.

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

While there are many similarities between a database running on SPARC or RISC and one running on x86 or x86-64, 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 to a standardized file format and then importing it into a new database. 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.

The choice of whether to use an open-source database such as MySQL or a proprietary database such as Oracle Database is driven by cost, application requirements, and business needs. If the same database vendor can be maintained on both platforms, the migration process is much simpler and straightforward.

Although it is challenging to move data between proprietary databases, most database vendors provide tools to assist data migration. For information about migrating from various proprietary databases to Oracle Database, see https://www.oracle.com/technetwork/products/migration/index-084442.html.

2.5 Verifying Applications

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.

The software described in this documentation is either in Extended Support or Sustaining Support. See https://www.oracle.com/us/support/library/enterprise-linux-support-policies-069172.pdf for more information. Oracle recommends that you upgrade the software described by this documentation as soon as possible.

Oracle Solaris Studio provides the Uncover and Tcov tools to track code coverage during testing, and the Discover tool to unearth difficult to detect code issues, such as memory access errrors and leaks. Alternatively, you can use open-source tools such as Gcov and Valgrind.

For information about using Uncover, Tcov, and Discover, see the Oracle Solaris Studio Documentation.

2.5.1 Using Gcov to Analyze Code Coverage

Gcov is an open-source code-coverage tool.

To use Gcov, perform the following steps:

1. Compile the code with the `-fprofile-arcs` and `-ftest-coverage` flags, for example:

   ```
   $ gcc -fprofile-arcs -ftest-coverage test.c
   ```

   The `-ftest-coverage` flag causes `gcc` to add instrumentation codes to the binary.

2. Run the instrumented binary and perform functional testing.

   Running the binary generates profile output. For each source file that you compiled with `-fprofile-arcs`, a `.gcda` profile output file is created in the object file directory.

3. Generate a report file based on the data that is stored in the profile output files:

   ```
   $ gcov test.c
   56.0% of 110 source lines executed in file test.c
   Creating test.c.gcov.
   ```

For more information about using `gcov`, see the `gcov(1)` manual page.

2.5.2 Using Valgrind to Detect Memory Access Errors and Leaks

Valgrind is a open-source memory access error and leak detection tool.

To use Valgrind, perform the following steps:

1. Compile the code with the `-g` flag, for example:

   ```
   $ gcc -g -O1 test.c
   ```

   An optimization level of 1 is generally faster than level 0, although it can cause incorrect line numbers to be reported.

   An optimization level higher than 1 can cause spurious uninitialised-value errors to be reported.

2. Use the `valgrind` as a wrapper for running the binary and perform stress testing:

   ```
   $ valgrind --leak-check=yes --log-file=valgrind.rpt a.out
   ```

   Memory access checking is enabled by default. The `--leak-check` option runs the memory leak detector when the binary exits. If you specify its value as `summary`, it reports how many leaks occurred. A value of `full` or `yes` displays the details of each individual leak.

For more information about using Valgrind, see the `valgrind(1)` manual page and the Valgrind Documentation at https://www.valgrind.org/docs/manual/index.html.

The software described in this documentation is either in Extended Support or Sustaining Support. See https://www.oracle.com/us/support/library/enterprise-linux-support-policies-069172.pdf for more information.
Oracle recommends that you upgrade the software described by this documentation as soon as possible.

Chapter 3 Operating System Considerations

Table of Contents

3.1 Storage Order and Alignment 9
3.2 Data Structures and Sizes 10
3.3 Compiler Options and Portability of Code 11
3.4 Byte Ordering 11
3.5 Data Conversion for Interoperability 12
3.5.1 Low-Level Code, Bit-Level Operations 12
3.6 System Call Mapping 12

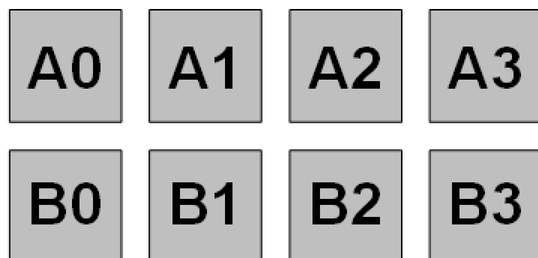
Although various flavors of Linux are available on SPARC or RISC platforms, most Linux deployments are on x86 or x86-64 systems. Migrating to Oracle Linux while remaining on an x86 or x86-64 system is a simple task because few processor-specific issues arise during this transition. On the other hand, specific measures must be taken when transitioning from a SPARC or RISC platform.

This chapter discusses the various aspects that must be considered during migration between platform architectures.

3.1 Storage Order and Alignment

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. Figure 3.1 illustrates how a correctly aligned integer value is stored in memory. The 4 bytes of the aligned integer `A` are stored as `A0`, `A1`, `A2`, and `A3` in memory. As this integer is stored with correct alignment, the processor can fetch the complete word in a single 32-bit bus cycle.

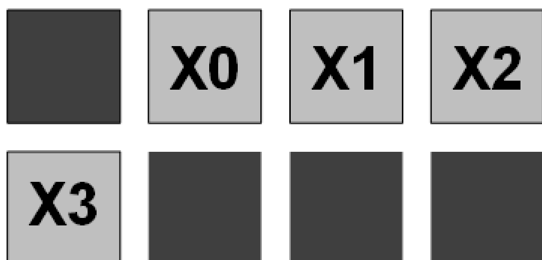
Figure 3.1 Correctly Aligned Integer Values in Memory



If the same processor now attempts to access an integer variable `X` at an unaligned address, it cannot perform the read in a single bus cycle. Figure 3.2 illustrates an incorrectly aligned integer value stored in memory.

The software described in this documentation is either in Extended Support or Sustaining Support. See https://www.oracle.com/us/support/library/enterprise-linux-support-policies-069172.pdf for more information.
Oracle recommends that you upgrade the software described by this documentation as soon as possible.

Figure 3.2 Incorrectly Aligned Integer Value in Memory



The processor has to issue two fetch instructions to read the complete misaligned integer and so takes twice as long as for an aligned 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.

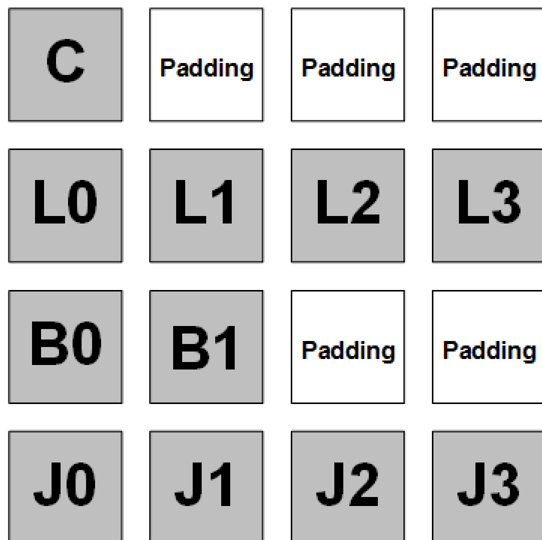
3.2 Data Structures and Sizes

To be able 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;
};
```

Figure 3.3 illustrates how `struct mydata` would be padded to align with 4-byte boundaries.

Figure 3.3 Memory Alignment and Padding of Structures



As 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`. Because of the padding, the addresses of the data in this structure are

The software described in this documentation is either in Extended Support or Sustaining Support. See https://www.oracle.com/us/support/library/enterprise-linux-support-policies-069172.pdf for more information. Oracle recommends that you upgrade the software described by this documentation as soon as possible.

evenly divisible by 4. This is called structure member alignment. Obviously, the size of the structure in memory grows as a consequence.

3.3 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, an application might send serialized data over a network. Avoiding byte padding can drastically improve the network utilization.

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

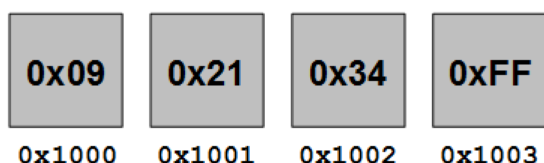
It should be clear by now that to be able to transfer the raw data from one platform and load it on another, the two platforms not only need to have fundamental types of the same size and of the same endianess, but they also need to be alignment-compatible. Otherwise, the positions of members inside the type, and even the size of the type itself, can differ. This is exactly what happens if the data corresponding to `mystruct` is moved between an x86 or x86-x64 system and a SPARC system, even though the types used in the structure are the same size.

3.4 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.

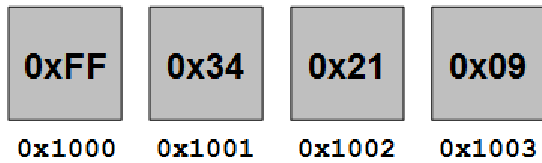
Figure 3.4 and Figure 3.5 show a representation of the hexadecimal value `0xFF342109`, where the number is stored at memory locations `0x1000` through `0x1003` on a little-endian machine and a big-endian system respectively.

Figure 3.4 Representation of 0xFF342109 on a Little-endian System



The software described in this documentation is either in Extended Support or Sustaining Support. See https://www.oracle.com/us/support/library/enterprise-linux-support-policies-069172.pdf for more information.
Oracle recommends that you upgrade the software described by this documentation as soon as possible.

Figure 3.5 Representation of 0xFF342109 on a Big-endian System



3.5 Data Conversion for Interoperability

The logic for converting between the big-endian and little-endian formats is actually quite straight forward. Figure 3.4 and Figure 3.5 show that you can convert one format to the other by swapping the appropriate bytes.

To account for communication over a network that has machines with diverse architectures, the data is usually 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. You can use conversion routines such as `ntohl()` and `htonl()` convert from network to host-byte order and from host to network-byte order respectively.

For member data in structure, union, or class objects, the structure members are aligned to the highest bytes of the size of any member to prevent performance penalties. In the following example, the size of `mystruct` is 8 bytes.

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

However, in the next example, the size of `mystruct` is 16 bytes as the `i` member is 8 bytes in size.

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

3.5.1 Low-Level Code, Bit-Level Operations

When migrating from a 32-bit application to a 64-bit application, bit-shifting operations can lead to errors. Untyped integral constants are assumed to be of type `unsigned int`. During shifting, this assumption can lead to unexpected truncation. For example, the maximum possible value of `i` in the following code sample is 31 because the implicit type of `1` is `unsigned int`.

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

To allow the shift to work correctly on a 64-bit system, use `1L` instead of `1`:

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

3.6 System Call Mapping

The Oracle Linux operating system follows the POSIX standard and provides well-defined system call interfaces. Most of the system calls available on Oracle Linux are also available on other POSIX-compliant

The software described in this documentation is either in Extended Support or Sustaining Support. See https://www.oracle.com/us/support/library/enterprise-linux-support-policies-069172.pdf for more information.
Oracle recommends that you upgrade the software described by this documentation as soon as possible.

operating systems, either as system calls or library functions (APIs). There can be some additional minor differences in system call implementation between platforms.

The following sections list potential implementation differences in the number of arguments, argument types, return values, return value types, or differences in the headers files that should be included. In addition, there are likely to be differences in the errno values that can be set if an error occurs, or in the signals and argument flags that are supported.

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

System Call	Possible Implementation Differences
#include <unistd.h> int access(const char *pathname, int mode);	Some operating systems require additional header files such as <sys/fcntl.h> to be included.
#include <unistd.h> int acct(const char *filename);	Some operating systems define a different argument type or do not require the <unistd.h> header file to be included.
#include <unistd.h> int brk(void *addr);	Some operating systems define a different argument type.
#include <unistd.h> int chown(const char *path, uid_t owner, gid_t group);	Some operating systems require additional header files such as <sys/types.h> to be included.
#include <sys/types.h> #include <sys/stat.h> #include <fcntl.h> int creat(const char *pathname, mode_t mode);	Some operating systems might not require <sys/types.h> or <sys/stat.h> to be included.
#define _ATFILE_SOURCE #include <fcntl.h> /* Definition of AT_* constants */ #include <unistd.h> int faccessat(int dirfd, const char *pathname, int mode, int flags);	Some operating systems do not require _ATFILE_SOURCE to be defined and might require <sys/fcntl.h> to be included instead of <fcntl.h>.
#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);	Some operating systems do not require _ATFILE_SOURCE to be defined and might not require <fcntl.h> to be included.

The software described in this documentation is either in Extended Support or Sustaining Support. See https://www.oracle.com/us/support/library/enterprise-linux-support-policies-069172.pdf for more information.
Oracle recommends that you upgrade the software described by this documentation as soon as possible.

System Call	Possible Implementation Differences
#include <unistd.h> int fchown(int *fd*, uid_t *owner*, gid_t *group*);	Some operating systems require additional header files such as <sys/types.h> to be included.
#define _ATFILE_SOURCE #include <fcntl.h> /* Definition of AT_* constants */ #include <unistd.h> int fchownat(int *dirfd*, const char **pathname*, uid_t *owner*, gid_t *group*, int *flags*);	Some operating systems do not require _ATFILE_SOURCE to be defined and might require <sys/types.h> to be included but not <fcntl.h>.
#include <unistd.h> #include <fcntl.h> int fcntl(int *fd*, int *cmd*, ... /* *arg* */);	Some operating systems require additional header files such as <sys/types.h> to be included or do not require <unistd.h>.
#include <unistd.h> pid_t fork(void);	Some operating systems include additional header files such as <sys/types.h>.
#include <sys/types.h> #include <sys/stat.h> #include <unistd.h> int fstat(int *fd*, struct stat **buf*);	Some operating systems require the <fcntl.h> header file to be included or do not require <sys/types.h> or <unistd.h>. References to dev_t objects might require the use of the major, minor, and makedev macros that are defined in <sys/sysmacros.h>.
#include <sys/vfs.h> /* or <sys/statfs.h> */ int fstatfs(int *fd*, struct statfs **buf*);	Some operating systems might require the <sys/statfs.h> header file to be included. <sys/vfs.h> and <sys/statfs.h> are equivalent on Oracle Linux.
#define _ATFILE_SOURCE #include <fcntl.h> /* Definition of AT_* constants */ int futimesat(int *dirfd*, const chat **pathname*, const struct timeval *times*[2]);	Some operating systems do not require _ATFILE_SOURCE to be defined and might include <sys/time.h> but not <fcntl.h>.
int getdents(unsigned int *fd*, struct linux_dirent **dirp*, unsigned int *count*);	On some operating systems, the argument types to getdents might be different and the <dirent.h> header file might also need to be included. getdents is not POSIX compliant. For POSIX compatibility when porting, consider using readdir instead.

The software described in this documentation is either in Extended Support or Sustaining Support. See https://www.oracle.com/us/support/library/enterprise-linux-support-policies-069172.pdf for more information.
Oracle recommends that you upgrade the software described by this documentation as soon as possible.

System Call	Possible Implementation Differences

`#include <sys/types.h>`
 	
`#include <unistd.h>`
 	
`int getgroups(int size, gid_t list[]);`
	On some operating systems, the argument types to `getdents` are different and the `<sys/types.h>` header file might not be need to be included.

`#include <sys/types.h>`
 	
`#include <unistd.h>`
 	
`pid_t getpid(void);`
	On some operating systems, the `<sys/types.h>` header file might not be need to be included.

`#include <sys/types.h>`
 	
`#include <unistd.h>`
 	
`pid_t getppid(void);`
	On some operating systems, the `<sys/types.h>` header file might not be need to be included.

`#include <sys/time.h>`
 	
`#include <sys/resource.h>`
 	
`int getrlimit(int resource, struct rlimit *rlim);`
	On some operating systems, the `<sys/time.h>` header file might not be need to be included.

`#include <sys/ioctl.h>`
 	
`int ioctl(int d, int request, ...);`
	Some operating systems require additional header files such as `<unistd.h>`, `<sys/types.h>`, or `<stropts.h>` to be included, and `<sys/ioctl.h>` might not be required.
	
	`ioctl` is not compatible between operating systems as command requests are usually specific to device drivers. Time and effort is required to produce equivalent behavior when migrating code that uses `ioctl` calls.

`#include <sys/klog.h>`
 	
`int klogctl(int type, char *bufp, int len);`
	`klogctl` is specific to Oracle Linux and is not POSIX compliant. On some operating systems, `klogctl` might not be implemented or the arguments and argument types to `klogctl` might be different.
	
	Time and effort is required to produce equivalent behavior when migrating code that uses `klogctl` calls.

`#include <unistd.h>`
 	
`int lchown(const char *path, uid_t owner, gid_t group);`
	Some operating systems require additional header files such as `<sys/types.h>` to be included.

`#define _ATFILE_SOURCE`
 	
`#include <fcntl.h> /* Definition of AT_* constants */`
 	
`#include <unistd.h>`
	Some operating systems do not require `_ATFILE_SOURCE` to be defined and might not require `<fcntl.h>` to be included.

The software described in this documentation is either in Extended Support or Sustaining Support. See https://www.oracle.com/us/support/library/enterprise-linux-support-policies-069172.pdf for more information.
Oracle recommends that you upgrade the software described by this documentation as soon as possible.

System Call	Possible Implementation Differences
int linkat(int *olddirfd*, const char **oldpath*, int *newdirfd*, const char **newpath*, int *flags*);	
#include <sys/types.h> #include <sys/stat.h> #include <unistd.h> int lstat(const char **path*, struct stat **buf*);	Some operating systems do not implement lstat or do not require the <sys/types.h> or <unistd.h> header files to be included.
#include <unistd.h> #include <sys/mman.h> int mincore(void **addr*, size_t *length*, unsigned char **vec*);	On some operating systems, the argument types to mincore might be different and the <sys/types.h> header file might also need to be included but not <unistd.h> or <sys/mman.h>.
#include <sys/stat.h> #include <sys/types.h> int mkdir(const char **pathname*, mode_t *mode*);	Some operating systems might not require <sys/types.h> to be included.
#define _ATFILE_SOURCE #include <fcntl.h> /* Definition of AT_* constants */ #include <sys/stat.h> int mkdirat(int *dirfd*, const char **pathname*, mode_t *mode*);	Some operating systems do not require _ATFILE_SOURCE to be defined and might not require <fcntl.h> to be included.
#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*);	Some operating systems might not require <sys/types.h>, <fcntl.h>, or <unistd.h> to be included.
#define _ATFILE_SOURCE #include <fcntl.h> /* Definition of AT_* constants */ #include <sys/stat.h> int mknodat(int *dirfd*, const char **pathname*, mode_t *mode*, dev_t *dev*);	Some operating systems do not require _ATFILE_SOURCE to be defined and might not require <fcntl.h> to be included.

The software described in this documentation is either in Extended Support or Sustaining Support. See https://www.oracle.com/us/support/library/enterprise-linux-support-policies-069172.pdf for more information.
Oracle recommends that you upgrade the software described by this documentation as soon as possible.

System Call	Possible Implementation Differences
#include <sys/mount.h> int mount(const char *source, const char *target, const char *filesystemtype, unsigned long mountflags,const void *data);	mount is not POSIX compliant. On some operating systems, the arguments and argument types to mount might be different and the <sys/types.h>, <sys/mntent.h>, or <sys/vmount.h> header files might also need to be included. Time and effort is required to produce equivalent behavior when migrating code that uses mount calls.
#include <sys/mman.h> int mprotect(const void *addr, size_t len, int prot);	On some operating systems, the arguments and argument types to mprotect might be different.
#include <sys/types.h> #include <sys/ipc.h> #include <sys/msg.h> int msgctl(int msqid, int cmd, struct msqid_ds *buf);	Some operating systems might not require <sys/types.h> or <sys/ipc.h> to be included.
#include <sys/types.h> #include <sys/ipc.h> #include <sys/msg.h> int msgget(key_t key, int msgflg);	Some operating systems might not require <sys/types.h> or <sys/ipc.h> to be included.
#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);	On some operating systems, the argument types to msgrcv might be different and the <sys/types.h> and <sys/ipc.h> header files might not need to be included.
#include <sys/types.h> #include <sys/ipc.h> #include <sys/msg.h> int msgsnd(int msqid, const void *msgp, size_t msgsz, int msgflg);	Some operating systems might not require <sys/types.h> and <sys/ipc.h> to be included.
#include <poll.h> int poll(struct pollfd *fds, nfds_t nfds, int timeout);	On some operating systems, the argument types to poll might be different.
#define _GNU_SOURCE	Some operating systems do not require _GNU_SOURCE to be defined.

The software described in this documentation is either in Extended Support or Sustaining Support. See https://www.oracle.com/us/support/library/enterprise-linux-support-policies-069172.pdf for more information.
Oracle recommends that you upgrade the software described by this documentation as soon as possible.

System Call	Possible Implementation Differences

`#include <poll.h>`
 	
`int ppoll(struct pollfd *fds, nfds_t nfds,const struct timespec *timeout, const sigset_t *sigmask);`	

`#include <unistd.h>`
 	
`int profil(unsigned short *buf, size_t bufsiz, size_t offset, unsigned int scale);`
	On some operating systems, the argument types and return value type of `profil` might be different or might require the `<time.h>` header file to be included but not `<unistd.h>`.

`#include <unistd.h>`
 	
`ssize_t readlink(const char *path, char *buf, size_t bufsiz);`
	On some operating systems, the argument types and return value type of `readlink` might be different and might require the `<symlink.h>` header file to be included instead of `<unistd.h>`.

`#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);`
	Some operating systems do not require `_ATFILE_SOURCE` to be defined and might not require the `<fcntl.h>` header file to be included.

`#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);`
	Some operating systems do not require `_ATFILE_SOURCE` to be defined, and might require `<unistd.h>` to be included but not `<fcntl.h>` or `<stdio.h>`.

`#include <unistd.h>`
 	
`void *sbrk(intptr_t increment);`
	Some operating systems define a different argument type.

`#include <sys/types.h>`
 	
`#include <sys/ipc.h>`
 	
`#include <sys/sem.h>`
 	
`int semop(int semid, struct sembuf *sops, unsigned nsops);`
	On some operating systems, the argument types for `semop` might be different or the `<sys/types.h>` and `<sys/ipc.h>` header files might not need to be included.

`#include <sys/types.h>`
 	
`#include <sys/ipc.h>`
 	
`#include <sys/sem.h>`
	On some operating systems, the argument types for `semtimedop` might be different or the `<sys/time.h>` header file might need to be included instead of `<sys/types.h>` and `<sys/ipc.h>`.

The software described in this documentation is either in Extended Support or Sustaining Support. See https://www.oracle.com/us/support/library/enterprise-linux-support-policies-069172.pdf for more information.
Oracle recommends that you upgrade the software described by this documentation as soon as possible.

System Call	Possible Implementation Differences
`int semtimedop(int semid, struct sembuf *sops, unsigned nsops, struct timespec *timeout);`	

`#include <grp.h>`
 	
`int setgroups(size_t size, const gid_t *list);`
	On some operating systems, the argument types to `setgroups` might be different, and the `<unistd.h>` header file might need to be included instead of `<grp.h>`.

`#include <unistd.h>`
 	
`int setpgid(pid_t pid, pid_t pgid);`
	Some operating systems might require the `<sys/types.h>` header file to be included.

`#include <unistd.h>`
 	
`pid_t setsid(void);`
	Some operating systems might require the `<sys/types.h>` header file to be included.

`#include <sys/ipc.h>`
 	
`#include <sys/shm.h>`
 	
`int shmctl(int shmid, int cmd, struct shmid_ds *buf);`
	Some operating systems might require the `<sys/types.h>` header file to be included or not require `<sys/ipc.h>`.

`#include <sys/ipc.h>`
 	
`#include <sys/shm.h>`
 	
`int shmget(key_t key, size_t size, int shmflg);`
	Some operating systems might require the `<sys/types.h>` header file to be included or not require `<sys/ipc.h>`.

`#include <signal.h>`
 	
`int sigaction(int signum, const struct sigaction *act, struct sigaction *oldact);`
	On some operating systems, the argument types to `sigaction` might be different.

`#include <signal.h>`
 	
`int sigaltstack(const stack_t *ss, stack_t *oss);`
	On some operating systems, the argument types to `sigaltstack` might be different.

`#include <signal.h>`
 	
`int sigprocmask(int how, const sigset_t *set, sigset_t *oldset);`
	On some operating systems, the argument types to `sigprocmask` might be different.

`#include <sys/types.h>`
 	
`#include <sys/stat.h>`
 	
`#include <unistd.h>`
 	
`int stat(const char *path, struct stat *buf);`
	Some operating systems might require the `<fcntl.h>` header file to be included or not require `<unistd.h>`.

`#include <sys/vfs.h> /* or <sys/statfs.h> */`	Some operating systems might require the `<sys/statfs.h>` header file to be included. `<sys/`

The software described in this documentation is either in Extended Support or Sustaining Support. See https://www.oracle.com/us/support/library/enterprise-linux-support-policies-069172.pdf for more information.
Oracle recommends that you upgrade the software described by this documentation as soon as possible.

System Call	Possible Implementation Differences
`int statfs(const char *path, struct statfs *buf);`	`vfs.h` and `<sys/statfs.h>` are equivalent on Oracle Linux.

`#include <time.h>`
 	
`int stime(time_t *t);`
	On some operating systems, the argument type to `stime` might be different or the `<unistd.h>` header file might need to be included instead of `<time.h>`.

`#define _ATFILE_SOURCE`
 	
`#include <fcntl.h> /* Definition of AT_* constants */`
 	
`#include <stdio.h>`
 	
`int symlinkat(const char *oldpath, int newdirfd, const char *newpath);`
	Some operating systems do not require `_ATFILE_SOURCE` to be defined and might require `<unistd.h>` to be included but not `<fcntl.h>` or `<stdio.h>`.

`int sysfs(int option);`
 	
`int sysfs(int option, const char *fsname);`
 	
`int sysfs(int option, unsigned int fs_index, char *buf);`
	On some operating systems, the supported arguments and argument types to `sysfs` might be different and the `<sys/fstyp.h>` or `<sys/fsid.h>` header files might need to be included.

`#include <sys/sysinfo.h>`
 	
`int sysinfo(struct sysinfo *info);`
	On some operating systems, the supported arguments and argument types to `sysfs` might be different and the `<sys/systeminfo.h>` header file might need to be included instead of `<sys/sysinfo.h>`.
	
	Time and effort is required to produce equivalent behavior when migrating code that uses `sysinfo` calls.

`int syslog(int type, char *bufp, int len);`	`syslog` is specific to Oracle Linux and is not POSIX compliant. On some operating systems, `syslog` might not be implemented or the arguments and argument types to `syslog` might be different. Time and effort is required to produce equivalent behavior when migrating code that uses `syslog` calls.

`#include <time.h>`
 	
`time_t time(time_t *t);`
	Some operating systems might also require the `<sys/types.h>` header file to be included.
	
	The method used to determine the time might depend on the system architecture.

`#include <sys/times.h>`
 	
`clock_t times(struct tms *buf);`
	Some operating systems might also require the `<limits.h>` header file to be included.

`#include <sys/utsname.h>`	Some operating systems might define `domainname` as part of the `utsname` structure.

The software described in this documentation is either in Extended Support or Sustaining Support. See https://www.oracle.com/us/support/library/enterprise-linux-support-policies-069172.pdf for more information.
Oracle recommends that you upgrade the software described by this documentation as soon as possible.

System Call	Possible Implementation Differences
`int uname(struct utsname *buf);`	

`#define _ATFILE_SOURCE`
`#include <fcntl.h>`
`int unlinkat(int dirfd, const char *pathname, int flags);`
	Some operating systems do not require `_ATFILE_SOURCE` to be defined and might require `<unistd.h>` to be included but not `<fcntl.h>`.

`#include <sys/types.h>`
`#include <unistd.h> /* libc[45] */`
`#include <ustat.h> /* glibc2 */`
`int ustat(dev_t dev, struct ustat *ubuf);`
	Some operating systems do not require the `<unistd.h>` or `<sys/types.h>` header files to be included.

`#include <unistd.h>` `int vhangup(void);`	On some operating systems, the return value type of `vhangup` might be different.

The software described in this documentation is either in Extended Support or Sustaining Support. See https://www.oracle.com/us/support/library/enterprise-linux-support-policies-069172.pdf for more information. Oracle recommends that you upgrade the software described by this documentation as soon as possible.

Chapter 4 Application Development Environment

Table of Contents

4.1 GNU Compiler Collection ... 23
4.2 Oracle Solaris Studio for Oracle Linux ... 23
4.3 Optimizing gcc Compilation ... 24
4.4 Open Source Software Libraries ... 25
4.5 Debugging Applications ... 25
4.6 Identifying Issues Using DTrace ... 26

When the set of commands, tools, libraries, and platform services are similar on the source and target platforms, the migration of an application development environment can be very smooth and the complexities of porting complexities are minimized. For example, the GNU Compiler Collection (GCC) and other GNU tools are available on the Oracle Solaris, Oracle Linux, and other Linux and UNIX platforms. A first step towards migration to Oracle Linux can be to move to using GCC development on the source platform.

Oracle has made the Oracle Solaris Studio 12.x compilers and tools available on the Oracle Linux, Oracle Solaris, and Red Hat Enterprise Linux platforms, which can help if you are porting applications from Oracle Solaris to Oracle Linux or Red Hat Enterprise Linux.

This chapter provides more information about these development environments.

4.1 GNU Compiler Collection

The GNU Compiler Collection (GCC) consists of utilities that you can use to compile programs, written in languages that include C, C++, Objective-C, Objective-C++, and Fortran. The GCC provides compilers such as `gcc` and `g++`, run-time libraries such as `libgcc`, `libstdc++`, and `libgfortran`, together with other useful tools.

The version of GCC in Oracle Linux 6 is based on the 4.4.x release series, about which you can find more information at http://gcc.gnu.org/gcc-4.4/.

The C compiler in GCC 4.4.x is largely compatible with the C99 ABI standard. For information about deviations from the C99 standard, see http://gcc.gnu.org/gcc-4.4/c99status.html.

For more information about compliance with language standards in GCC, see http://gcc.gnu.org/onlinedocs/gcc/Standards.html#Standards.

4.2 Oracle Solaris Studio for Oracle Linux

Oracle Solaris Studio is a comprehensive C, C++, and Fortran tool suite 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. The tools enable easier creation of parallel and concurrent software applications for such platforms. The compilers, tools, and libraries shipped with Oracle Solaris Studio are engineered to make applications run optimally on Oracle platforms.

Oracle Solaris Studio includes a number of tools that can help you to isolate difficult-to-detect code issues:

- Performance Analyzer allows you to analyze application performance, to determine which parts of a program are candidates for improvement, and to help identify performance hotspots.

The software described in this documentation is either in Extended Support or Sustaining Support. See https://www.oracle.com/us/support/library/enterprise-linux-support-policies-069172.pdf for more information.
Oracle recommends that you upgrade the software described by this documentation as soon as possible.

- Thread Analyzer allows you to detect code issues in multithreaded programs such as data races and deadlock conditions.
- Discover allows you to detect programming errors related to the allocation and use of program memory at runtime, such as:
  - Reading from and writing to unallocated memory.
  - Freeing incorrect memory blocks.
  - Attempting to use freed memory.
  - Accessing memory beyond allocated array bounds.
  - Memory leaks.
  - Accessing uninitialized memory.
- Uncover allows you to measure the code coverage of user applications by displaying information about the areas of an application that are exercised during testing.

The C compiler (`cc`) in Oracle Solaris Studio is fully compliant 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 and 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 adhere to IEEE 754. If the application to be ported does not have strict requirements for standards adherence, you can improve performance further by using optimization flags that create higher-performance binaries. For example, the C compiler option `-fns` permits nonstandard floating-point truncation and the `-fast` macro (which implies `-fns`) generates an optimized binary for specific targeted hardware.

For more information about Oracle Solaris Studio, see the Oracle Solaris Studio Documentation.

4.3 Optimizing gcc Compilation

The `-O` *`level`* option to `gcc` turns on compiler optimization, when the specified value of *`level`* has the following effects:

`0`
: The default reduces compilation time and has the effect that debugging always yields the expected result. This level is equivalent to not specifying the `-O` option at all. However, a number of optimization options are still enabled, for example: `-falign-loops`, `-finline-functions-called-once`, and `-fmove-loop-invariants`.

`1`
: The compiler attempts to reduce both the size of the output binary code and the execution speed, but it does not perform optimizations that might substantially increase the compilation time.

`2`
: The compiler performs optimizations that do not require a tradeoff of space for speed. Compared to level 1, level 2 optimization improves the performance of the output binary but it also increases the compilation time.

`3`
: The compiler turns on the `-fgcse-after-reload`, `-finline-functions`, `-fipa-cp-clone`, `-fpredictive-commoning`, `-ftree-`

The software described in this documentation is either in Extended Support or Sustaining Support. See https://www.oracle.com/us/support/library/enterprise-linux-support-policies-069172.pdf for more information.
Oracle recommends that you upgrade the software described by this documentation as soon as possible.

	`vectorize`, and `-funswitch-loops` options, which require a tradeoff of space for speed, in addition to the level 2 optimizations.
`s`	The compiler optimizes to reduce the size of the binary instead of execution speed.

If you do not specify an optimization option, `gcc` attempts to reduce the compilation time and to make debugging always yield the result expected from reading the source code. If you enable optimization, the compiler tries to improve performance, to reduce the size of the output binary, or both, but compilation takes longer and you can lose the ability to debug the program effectively. If you compile several source files together to a single output binary file, the compiler uses information that it gathers from all of the source files when compiling each individual source file.

You can use the following command to find out which optimization options are enabled at a specified optimization level:

```
$ gcc -c -Q -Olevel --help=optimizers
```

To improve the speed of compilation, you can specify the `-pipe` option, which instructs `gcc` to use pipes rather than temporary files for communication between the various stages of compilation.

Taking advantage of hardware properties specific to target platforms can result in a significant performance improvement. By default, GCC compiles code that is optimized for the most processors. However, you can use the `-mtune` and `-march` options with `gcc` to optimize instruction scheduling and instruction set selection respectively. Specifying an architecture with `-march` implicitly selects the value of `-mtune` unless you specify a value explicitly. Your program can still run, albeit probably not optimally, if you specify an incorrect value for `-mtune`, but it is likely to fail if you specify an incorrect value for `-march`. For more information, see the `gcc(1)` manual page and the *GCC 4.4.4 Manual: Hardware Models and Configurations*.

4.4 Open Source Software Libraries

Most popular open source and GNU software is either already available on Oracle Linux as RPM packages that you can install by using `yum`. Ported binaries as well as source code are also available from open-source repositories.

4.5 Debugging Applications

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

- Kernel-mode debuggers, which allow you to debug code that runs in the privilege mode of the CPU. Examples of kernel modules include file system kernel modules, device drivers, and so on.

  Adding `printk()` statements to code can be a simple and effective way to locate problems. The `crash` utility allows you to examine the live kernel, user processes, system dumps, memory leaks, and application core dumps. The kernels that are shipped with Oracle Linux are stripped so you need to build a non-stripped version of the kernel if you want to use the `kdb` or `kdbg` debuggers on it.

- User-mode debuggers, which allow you to debug code that runs in the non-privilege mode of the CPU. User applications execute in user mode and the applications use system calls to interface to the kernel. Code that runs in kernel mode code has a different address space from code that runs in user mode.

  You can use also use GDB for source-level symbolic debugging and execution of programs written in ANSI C and C++, and other programming languages are partially supported. Useful features of GDB

The software described in this documentation is either in Extended Support or Sustaining Support. See
https://www.oracle.com/us/support/library/enterprise-linux-support-policies-069172.pdf for more information.
Oracle recommends that you upgrade the software described by this documentation as soon as possible.

include stepping through programs one source line or one machine instruction at a time. In addition to simply viewing the operation of the program, you can manipulate variables and evaluate and display a wide range of expressions. You can debug a program, a running process, or a core file. To use GDB effectively, you build your source code with the `-g` compiler flag to create a debug binary. Full debugging information also requires that you select no compiler optimization. Only partial debugging support is available for optimized code. GDB 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. You can also navigate between threads, suspend or step a thread, and display the stack and any locks.

4.6 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 that are 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 on a production system, assuming that the user is aware of the possible overhead and impact of running the script.

For additional information about using DTrace on Oracle Linux, see *Oracle® Linux: DTrace Guide* and *Oracle® Linux: DTrace Tutorial*.

The software described in this documentation is either in Extended Support or Sustaining Support. See https://www.oracle.com/us/support/library/enterprise-linux-support-policies-069172.pdf for more information.
Oracle recommends that you upgrade the software described by this documentation as soon as possible.

Chapter 5 Threads and Multiprocessing

Table of Contents

5.1 POSIX Compliance 27
5.2 Threading Model 27
5.3 Differences Between Implementations of Pthreads 27
5.4 Thread Attributes 28
5.5 Signals in Threaded Applications 28
5.6 OpenMP Support 29
5.7 Auto Parallelization and Compile-Time Optimizations 29
5.8 Using the Thread Analyzer 29

This chapter provides information about factors that you should take into consideration when porting threaded applications in a multiprocessor environment.

5.1 POSIX Compliance

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

5.2 Threading Model

The GNU C library (glibc) implements Pthreads as the Native POSIX Threads Library (NPTL), where each thread maps to a kernel scheduling entity. The original Pthreads implementation Linux threads is not supported from glibc 2.4 onward. If an application depends on Linux threads being present, you must modify it to use NPTL.

Compared with LinuxThreads, NPTL provides closer conformance to the requirements of the POSIX.1 specification and better performance for large numbers of threads. NPTL has been available from glibc 2.3.2 onward and requires features that are present in the 2.6 and later kernels.

For glibc 2.3.2 and later, you can use the `getconf` command to determine the threading implementation:

```
# getconf GNU_LIBPTHREAD_VERSION
NPTL 2.12
```

With older glibc versions, use a command such as the following to determine the default threading implementation:

```
# $( ldd /bin/ls | grep libc.so | awk '{print $3}' ) | egrep -i 'threads|nptl'
 Native POSIX Threads Library by Ulrich Drepper et al
```

5.3 Differences Between Implementations of Pthreads

Minor differences in the implementations of Pthreads are largely due to edge cases that were either left unspecified by the standard or by permitted implementation dependences, for example:

- Pthreads in Oracle Solaris has a priority scheduling attribute that does not exist in NPTL on Oracle Linux.

The software described in this documentation is either in Extended Support or Sustaining Support. See https://www.oracle.com/us/support/library/enterprise-linux-support-policies-069172.pdf for more information.
Oracle recommends that you upgrade the software described by this documentation as soon as possible.

- In the POSIX standard, all threads are assumed to be part of same process and so share a common nice value. NPTL threads on Oracle Linux are in effect separate processes and do not necessarily share a common nice value.

The following table describes aspects of Pthread programming in Oracle Linux that can differ from other operating systems.

Function	Implementation Comments
`int pthread_atfork(void (*prepare)(void), void (*parent)(void), void (*child)(void));`	The *parent* and *child* fork handlers are called in the order in which they were established by calls to `pthread_atfork()`. The *prepare* fork handlers are called in the opposite order.

`int pthread_attr_destroy(pthread_attr_t *attr);`
 	
`int pthread_attr_init(pthread_attr_t *attr);`
	These functions always succeed. For portability, applications should handle a possible error return.

`int pthread_cancel(pthread_t thread);`
 	
`void pthread_cleanup_push(void (*routine)(void *), void *arg);`
 	
`void pthread_cleanup_pop(int execute)`
	NPTL implements thread cancellation by using the first real-time signal (signal 32).

`int pthread_create(pthread_t *thread, const pthread_attr_t *attr, void *(*start_routine) (void *), void *arg);`	The new thread inherits copies of the calling thread's capability sets and CPU affinity mask.

5.4 Thread Attributes

To list thread attributes on Oracle Linux, compile and run the C source program that is listed on the `pthread_attr_init(3)` manual page, for example:

```
$ gcc thread.c -pthread
$ ./a.out
Thread attributes:
 Detach state        = PTHREAD_CREATE_JOINABLE
 Scope               = PTHREAD_SCOPE_SYSTEM
 Inherit scheduler   = PTHREAD_INHERIT_SCHED
 Scheduling policy   = SCHED_OTHER
 Scheduling priority = 0
 Guard size          = 4096 bytes
 Stack address       = 0x7f8f06da6000
 Stack size          = 0x801000 bytes
```

5.5 Signals in Threaded Applications

The Pthreads implementation is process-centric on Oracle Linux. As far as signals are concerned, each thread is viewed as an independent process. The kernel does not differentiate between threads and processes. Within a single process, the threads share many resources such as virtual memory, file descriptors, and global variables. As a result, multithreaded applications might not handle signals in a POSIX-compliant way. Signals are directly sent to individual threads and not to the process as a whole.

The software described in this documentation is either in Extended Support or Sustaining Support. See https://www.oracle.com/us/support/library/enterprise-linux-support-policies-069172.pdf for more information.
Oracle recommends that you upgrade the software described by this documentation as soon as possible.

The POSIX standard specifies that a signal that is 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.

5.6 OpenMP Support

The `gcc` and Oracle Solaris Studio compilers can look for OpenMP `pragma` directives in the source code in order to build a parallel version of the application. Similar to automatic parallelization, the compiler does the additional work so that you do not have to manage the threads. OpenMP represents an incremental approach to parallelization with potentially fine granularity. OpenMP allows you to set directives around specific loops to be optimized through threading while leaving other loops untouched. The advantage of this approach is that you can derive a serial version and a parallel version of the application from the same code, which can be helpful for debugging.

To instruct the `gcc` compiler to recognize OpenMP directives, specify the `-fopenmp` flag.

Several compiler flags are available in Oracle Solaris Studio to support OpenMP. To instruct the compiler to recognize OpenMP directives, specify the `-xopenmp` flag. You can also set the `-xvpara` flag to report potential parallelization issues, and the `-loopinfo` flag to display information about which loops are parallelized.

At run time, set the `OMP_NUM_THREADS` environment variable to specify the number of processors that are available to the program.

5.7 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 Oracle Solaris Studio compilers provide mechanisms that let applications run multiple threads without requiring you to specify how. Within legacy code, execution loops usually present opportunities for optimization where repetitive serial operations can be divided into multiple, independent execution threads. Several compiler flags govern automatic parallelization behavior:

`-xautopar` Instructs the compiler to perform auto parallelization of loops.

`-xloopinfo` Outputs information about the loops that the compiler has parallelized.

`-xreduction` Instructs the compiler to parallelize reduction operations that take a range of values and output a single value, such as summing all the values in an array.

5.8 Using the Thread Analyzer

The Oracle Solaris Studio Thread Analyzer is an advanced tool for application optimization that is designed to help you detect and analyze race conditions and deadlocks in multithreaded applications.

Data races can cause incorrect or unpredictable results, and they can occur arbitrarily far way from where a problem appears 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.

The software described in this documentation is either in Extended Support or Sustaining Support. See https://www.oracle.com/us/support/library/enterprise-linux-support-policies-069172.pdf for more information. Oracle recommends that you upgrade the software described by this documentation as soon as possible.

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 where more than two threads are involved).

To instrument the source code for the detection of data race and deadlock issues, first compile the code with the `-xinstrument=datarace` flag, It is recommended that you also set the `-g` flag and that you specify no optimization level to help ensure that information about line numbers and call stacks is returned correctly.

You then execute the resulting application code using the `collect -r race` or `collect -r deadlock` command, which collect runtime information for the detection of data races or deadlocks during the execution of the process.

```
$ collect -r race app params
$ collect -r deadlock app params
```

Finally, load the results of the experiment into the Thread Analyzer to identify any data race or deadlock conditions.

The software described in this documentation is either in Extended Support or Sustaining Support. See https://www.oracle.com/us/support/library/enterprise-linux-support-policies-069172.pdf for more information.
Oracle recommends that you upgrade the software described by this documentation as soon as possible.

Chapter 6 Migrating Device Drivers

Table of Contents

6.1 Considerations for Porting Device Drivers 31
6.2 Reading and Writing Data from or to User Space 32
6.3 About Handling Access to Shared Resources 32
6.4 About the Bus Model 33
6.5 About Character Device Drivers 35
6.6 About Block Device Drivers 37
6.7 About Network Device Drivers 38
6.8 About USB Device Drivers 41
6.9 About the Sysfs File System 42
6.10 Loading Device Drivers as Kernel Modules 42

Device driver software is tightly coupled to the operating system on which it is running. To be able to develop a device driver, you must have a thorough knowledge of the operating system architecture and internals. When writing a device driver, you need to study the operating system (kernel) interfaces that your driver will use to access the hardware.

This chapter provides an overview of how Oracle Linux device drivers are usually implemented and contrasts this approach with that usually encountered on UNIX-like operating systems.

For more information about how to write Device Drivers for Linux systems, see *Linux Device Drivers* by Jonathan Corbet, Alessandro Rubini, and Greg Kroah-Hartman.

6.1 Considerations for Porting Device Drivers

The porting of device drivers between platforms can be complicated for various reasons:

- Even if the kernel data structures have the same name, additional members might provide enhanced functionality, some members might be missing, or the type or size of members might be different.
- Even if a kernel function has the same name, the calling syntax or return syntax might be different or there might be different prerequisites or side effects when calling the function.
- Drivers might not use existing kernel functions, but instead rely on undocumented side effects that are unique to the operating system.
- Architectural differences, such as word size and the size of data types such as `short`, `int`, and `long` can result in porting issues. The sizes of an address pointer and `int` are also not necessarily the same. Opaque data types such as `dev_t` and `atomic_t` might also not be the same size on the source and target systems so it is not safe to convert them to a non-opaque data type. The default signedness of `char` can also vary between platforms.
- Some architectures permit the use of unaligned data while other do not. For portability, all types should be naturally aligned on memory addresses that are multiples of their size.
- Structure padding might differ on source and target systems, which can cause porting issues if the code makes assumptions about how a structure is padded.
- The endianness of the system architecture, as defined in `<asm/byteorder.h>` on Linux systems, can cause porting problems if data structures are passed between systems, or if assumptions are made about the default byte order when processing data.

The software described in this documentation is either in Extended Support or Sustaining Support. See https://www.oracle.com/us/support/library/enterprise-linux-support-policies-069172.pdf for more information.
Oracle recommends that you upgrade the software described by this documentation as soon as possible.

- The number of timer interrupts per second and the internal representation of system time can vary between platforms. You should use the value of `HZ` (the number of timer interrupts per second, defined via `CONFIG_HZ` in `<config/auto.conf>`) to scale time units. For example, the time in seconds from booting the system can be found by dividing the value of the global variable `jiffies` by `HZ`.
- The memory page size can differ between platforms, and some architectures can support more than one page size.
- Different architectures can have different process ordering rules for controlling the order in which memory reads and writes happen. You can use the `rmb()`, `wmb()`, and `mb()` methods to implement barriers on reordering of memory reads and writes.
- Code that was not written for an SMP system must be modified to use locking where appropriate to protect critical sections where shared data is updated or to protect against kernel preemption if it is to run on an SMP system. The methods that are provided for locking might also differ between platforms and there might be different constraints on when certain types of locking can or should be used with driver methods and helper routines.
- When mapping and unmapping page structures to kernel address space, the `kmap()` and `kunmap()` functions should be used to allow for systems where high memory is not permanently mapped.
- Architecture-specific code might be scattered throughout the driver. Isolating the portable code from non-portable code can help reduce porting issues.

6.2 Reading and Writing Data from or to User Space

You can use the `copy_from_user()` and `copy_to_user()` functions to move data between kernel space and user space. Alternatively, when moving one, two, four, or eight bytes of data, you can use either `put_user()` and `get_user()` or `access_ok()` to validate the user-space address followed by either `__put_user()` or `__get_user()`.

If user programs require direct access to device memory, you can use the `mmap()` system call, which maps device memory directly into user space. For example, the X server uses `mmap()` to write to video adapter memory and PCI devices usually memory map their control registers to improve performance. A limitation is that the area being mapped has to be a multiple of `PAGE_SIZE` and start at a physical memory address that is also a multiple of `PAGE_SIZE`.

The direct I/O `get_user_pages()` function allows drivers to use user-space buffers without the overhead of copy data between user space and kernel space. However, most block and network drivers do not need to implement direct I/O as the hardware abstraction built into the kernel can use direct I/O when required. A driver that uses direct I/O to perform DMA operations will typically set up a scatter/gather list in the buffer. When using `get_user_pages()`, you must also use `down_read()` and `up_read()` to set up a read-mode semaphore around the call.

An alternative to using a direct I/O buffer that also supports asynchronous I/O is to use the `aio_read()` and `aio_write()` methods to read or write data. When the read or write operation completes, the driver informs the kernel by calling `aio_complete()`.

6.3 About Handling Access to Shared Resources

Multiple threads of execution can operate simultaneously on shared kernel data structures. As a result, you must ensure that such access is serialized. You can use spinlocks and mutexes to control access to critical sections, which are areas of code that contain shared resources.

The software described in this documentation is either in Extended Support or Sustaining Support. See https://www.oracle.com/us/support/library/enterprise-linux-support-policies-069172.pdf for more information.
Oracle recommends that you upgrade the software described by this documentation as soon as possible.

- A spinlock allows only one thread at a time to access a critical section. Another thread that wants to access the critical section continues to spin until the thread exits that has access. You can use the `spin_lock()` function to obtain a spinlock if the code does not run in either hardware or software interrupt context.

```
#include <linux/spinlock.h>

spinlock_t spinlock = SPIN_LOCK_UNLOCKED;

spin_lock(&spinlock);
    /* Critical section */
spin_unlock(&spinlock);
```

  If the code runs in either hardware or software interrupt context, use a spinlock function that disables interrupts, such as `spin_lock_irq()` or `spin_lock_irqsave()`. Finally, the `spin_lock_bh()` function disables software interrupts while leaving hardware interrupts enabled. Each of these functions is paired with a correspondingly named unlock function.

- A mutex puts a thread to sleep until it can access the critical section.

```
#include <linux/mutex.h>

static DEFINE_MUTEX(mutex);

mutex_lock(&mutex);
    /* Critical section */
mutex_unlock(&mutex);
```

  This example uses a statically defined mutex. You can use `mutex_init()` to create a dynamic mutex.

It is better to use mutexes than spinlocks when the wait time is expected to be more than two context switches. However, inside an interrupt handler, you must always use a spinlock because a mutex can put a thread to sleep. Conversely, if the critical section needs to sleep, you must use a mutex as a thread must not be scheduled, preempted, or put to sleep after it has acquired a spinlock.

Note

The mutex interface was introduced as an alternative to the semaphore interface. Semaphores allow simultaneous access by a specified number of threads to a critical section, but this feature is almost never used.

When a thread can only either read from or write to a critical region, there are also the `read_lock()` and `write_lock()` functions that you can use to implement reader or writer spin locks. A writer spinlock blocks access to all other writing or reading threads. A reader spinlock blocks access to other writing threads but allows access from other reading threads.

As an alternative to spinlocks and mutexes, you can use the 32 and 64-bit atomic integer types (`atomic_t` and `atomic64_t`) and perform operations such as incrementing counters and setting bit masks.

Finally, if a single writer thread maintains a data structure that is always seen consistently by a reader thread, you can use a lock-less circular buffer such as the `kfifo` structure that is defined in `<linux/kfifo.h>`. Network drivers typically use such buffers to exchange data with the adapter.

6.4 About the Bus Model

The device driver model in Oracle Linux uses global data structures to present data and control operations to the bus drivers for bridges and other devices. This uniform data model for describing a bus and its

The software described in this documentation is either in Extended Support or Sustaining Support. See https://www.oracle.com/us/support/library/enterprise-linux-support-policies-069172.pdf for more information.
Oracle recommends that you upgrade the software described by this documentation as soon as possible.

devices defines the bus attributes and callbacks for bus probing, device discovery, shutdown, power management, and other control operations. The model supports plug and play, power management, and hot plug functionality specified by the Advanced Configuration and Power Interface (ACPI) model, which applies to most devices in a modern x86 or x86-64 architecture system. A device driver registers its driver object with the bus subsystem. It can then use the vendor and product identifiers to determine if the hardware is present.

The kernel defines a common data structure for each bus that can be accessed by individual layers of a bus or by individual device drivers.

For example, the `pci_bus` data structure represents a PCI bus:

```
struct pci_bus {
        struct list_head node;          /* node in list of buses */
        struct pci_bus  *parent;        /* parent bus this bridge is on */
        struct list_head children;      /* list of child buses */
        struct list_head devices;       /* list of devices on this bus */
        struct pci_dev  *self;          /* bridge device as seen by parent */
        struct list_head slots;         /* list of slots on this bus */
        struct resource *resource[PCI_BRIDGE_RESOURCE_NUM];
        struct list_head resources;     /* address space routed to this bus */

        struct pci_ops  *ops;           /* configuration access functions */
        void            *sysdata;       /* hook for sys-specific extension */
        struct proc_dir_entry *procdir; /* directory entry in /proc/bus/pci */

        unsigned char   number;         /* bus number */
        unsigned char   primary;        /* number of primary bridge */
        unsigned char   secondary;      /* number of secondary bridge */
        unsigned char   subordinate;    /* max number of subordinate buses */
        unsigned char   max_bus_speed;  /* enum pci_bus_speed */
        unsigned char   cur_bus_speed;  /* enum pci_bus_speed */

        char            name[48];

        unsigned short  bridge_ctl;     /* manage NO_ISA/FBB/et al behaviors */
        pci_bus_flags_t bus_flags;      /* Inherited by child busses */
        struct device           *bridge;
        struct device           dev;
        struct bin_attribute    *legacy_io; /* legacy I/O for this bus */
        struct bin_attribute    *legacy_mem; /* legacy mem */
        unsigned int            is_added:1;
};
```

The `pci_slot` structure represents a slot on a PCI bus:

```
struct pci_slot {
        struct pci_bus *bus;            /* The bus this slot is on */
        struct list_head list;          /* node in list of slots on this bus */
        struct hotplug_slot *hotplug;   /* Hotplug info (migrate over time) */
        unsigned char number;           /* PCI_SLOT(pci_dev->devfn) */
        struct kobject kobj;
};
```

The `pci_dev` structure defines each device on a PCI bus:

```
struct pci_dev {
        struct list_head bus_list;      /* node in per-bus list */
        struct pci_bus  *bus;           /* bus this device is on */
        struct pci_bus  *subordinate;   /* bus this device bridges to */

        void            *sysdata;       /* hook for sys-specific extension */
        struct proc_dir_entry *procent; /* device entry in /proc/bus/pci */
```

The software described in this documentation is either in Extended Support or Sustaining Support. See https://www.oracle.com/us/support/library/enterprise-linux-support-policies-069172.pdf for more information.
Oracle recommends that you upgrade the software described by this documentation as soon as possible.


```
        struct pci_slot *slot;          /* Physical slot this device is in */

        unsigned int    devfn;          /* encoded device & function index */
        unsigned short  vendor;
        unsigned short  device;
        unsigned short  subsystem_vendor;
        unsigned short  subsystem_device;
        unsigned int    class;          /* 3 bytes: (base,sub,prog-if) */
        ...
        struct pci_driver *driver;      /* which driver has allocated this device */
        ...
        pci_channel_state_t error_state;        /* current connectivity state */
        struct  device  dev;            /* Generic device interface */

        int             cfg_size;       /* Size of configuration space */

        /*
         * Instead of touching interrupt line and base address registers
         * directly, use the values stored here. They might be different!
         */
        unsigned int    irq;
        struct resource resource[DEVICE_COUNT_RESOURCE]; /* I/O and memory regions
                                                            & expansion ROMs */
        resource_size_t fw_addr[DEVICE_COUNT_RESOURCE]; /* FW-assigned addr */
        ...
        pci_dev_flags_t dev_flags;
        atomic_t        enable_cnt;     /* pci_enable_device has been called */
        /* Several lines omitted */
};
```

Only some of the more important structure members are shown here.

There is one `pci_dev` structure for each combination of slot and function number. Every PCI device in a system, including PCI-PCI and PCI-ISA bridge devices, is represented by a `pci_dev` structure.

The `struct device dev` member of `pci_dev` represents the generic interface for a device. A PCI bus layer can access the members of this structure. However, individual PCI device drivers do not usually need to be able to access the structure. This abstraction prevents downstream drivers from breaking if a member of the structure is altered or removed. Only the code for the bus layer need be modified.

As with other bus-based subsystems, a PCI device driver registers its driver object with the PCI subsystem. It can then use vendor and device identifiers to determine if the hardware is present.

6.5 About Character Device Drivers

Character device drivers support devices that handle variable rather than fixed amounts of data, and which do not access physically addressable storage media or support file system access. Keyboards, mice, and video cards are examples of devices that might use character devices, although USB keyboards and mice require the use of USB device drivers.

The implementation of most character device drivers is very similar on both Linux and UNIX operating systems, with most differences arising from the structures that are used to define a driver's file operations.

The `file_operations` structure for a character device, defined in `<linux/fs.h>`, contains a set of method pointers that specify how the system interacts with the device via the device files under `/dev` when using system calls such as `open()`, `read()`, `write()`, and `ioctl()`.

```
struct file_operations {
        struct module *owner;
        loff_t (*llseek) (struct file *, loff_t, int);
```

The software described in this documentation is either in Extended Support or Sustaining Support. See https://www.oracle.com/us/support/library/enterprise-linux-support-policies-069172.pdf for more information.
Oracle recommends that you upgrade the software described by this documentation as soon as possible.


```
        ssize_t (*read) (struct file *, char __user *, size_t, loff_t *);
        ssize_t (*write) (struct file *, const char __user *, size_t, loff_t *);
        ssize_t (*aio_read) (struct kiocb *, const struct iovec *, unsigned long, loff_t);
        ssize_t (*read_iter) (struct kiocb *, struct iov_iter *, loff_t);
        ssize_t (*aio_write) (struct kiocb *, const struct iovec *, unsigned long, loff_t);
        ssize_t (*write_iter) (struct kiocb *, struct iov_iter *, loff_t);
        int (*readdir) (struct file *, void *, filldir_t);
        unsigned int (*poll) (struct file *, struct poll_table_struct *);
        long (*unlocked_ioctl) (struct file *, unsigned int, unsigned long);
        long (*compat_ioctl) (struct file *, unsigned int, unsigned long);
        int (*mmap) (struct file *, struct vm_area_struct *);
        int (*open) (struct inode *, struct file *);
        int (*flush) (struct file *, fl_owner_t id);
        int (*release) (struct inode *, struct file *);
        int (*fsync) (struct file *, int datasync);
        int (*aio_fsync) (struct kiocb *, int datasync);
        int (*fasync) (int, struct file *, int);
        int (*lock) (struct file *, int, struct file_lock *);
        ssize_t (*sendpage) (struct file *, struct page *, int, size_t, loff_t *, int);
        unsigned long (*get_unmapped_area)(struct file *, unsigned long, unsigned long,
                       unsigned long, unsigned long);
        int (*check_flags)(int);
        int (*flock) (struct file *, int, struct file_lock *);
        ssize_t (*splice_write)(struct pipe_inode_info *, struct file *, loff_t *,
                                size_t, unsigned int);
        ssize_t (*splice_read)(struct file *, loff_t *, struct pipe_inode_info *,
                               size_t, unsigned int);
        int (*setlease)(struct file *, long, struct file_lock **);
        long (*fallocate)(struct file *file, int mode, loff_t offset,
                          loff_t len);
};
```

A simple character device driver might only need to implement a subset of the functions defined for this structure, for example:

```
struct file_operations driver_fileops = {
    .owner   = THIS_MODULE,
    .llseek  = driver_llseek,
    .read    = driver_read,
    .write   = driver_write,
    .ioctl   = driver_ioctl,
    .open    = driver_open,
    .release = driver_release,
};
```

As well as the methods, it is usual to define `module_init()` initialization and `module_exit()` cleanup routines for the driver that are called when a driver is loaded and unloaded. These routines should call `register_chrdev()` and `unregister_chrdev()` to register and unregister the device major number for the driver.

Most character device drivers declare an interrupt handler to accept incoming data asynchronously from the device. When initializing the driver, use `request_irq()` to install the interrupt handler, remembering to specify shared interrupts by setting the `SA_SHIRQ` bit in the *`flags`* argument as well as a unique *`dev_id`* argument that the handler can use to identify interrupts that it should process. Cleanup should call `free_irq()` to unregister the handler. Alternatively, a driver can use polling to check for incoming data, although this is uncommon.

The `file` structure for a character device, defined in `<linux/fs.h>`, specifies the kernel-space data structure that the kernel creates when the device is opened.

The most important members of the `file` structure are:

The software described in this documentation is either in Extended Support or Sustaining Support. See https://www.oracle.com/us/support/library/enterprise-linux-support-policies-069172.pdf for more information.
Oracle recommends that you upgrade the software described by this documentation as soon as possible.

f_flags	File control flags that indicate how device operations should behave, for example, non-blocking reads.
f_mode	The mode, which indicates whether the file is readable and writable.
f_op	The operation associated with a file, usually depending on the minor device number.
f_pos	The current offset in the file for reading or writing.
private_data	Allows data about the device to be retained between system calls until the device is released.

To obtain the major and minor numbers from a device's inode, use the following macros that are also defined in <linux/fs.h>:

```
unsigned iminor(struct inode *inode);
unsigned imajor( struct inode *inode);
```

6.6 About Block Device Drivers

Block device drivers support devices that handle fixed rather than variable amounts of data and access physically addressable storage media or support file system access. Hard disks and flash memory are common types of block devices.

The implementation of block device drivers is likely to differ in many respects between Linux and UNIX operating systems, with most differences arising from the functions that the driver uses to handle block requests, the preferred use of the getgeo() (get geometry) method instead of an ioctl() method to support partitioning tools, and the use of the kernel block device interface (blkdev) to handle block I/O operations. The *driver_strategy()* routine that UNIX block device drivers use to handle the reading or writing of data from or to a device is named *driver_request()* on Linux systems.

The block_device_operations structure for a block device, defined in <linux/blkdev.h>, contains a set of method pointers that specify how the system interacts with the device via the device files under /dev when using system calls such as open() and ioctl().

```
struct block_device_operations {
        int (*open) (struct block_device *, fmode_t);
        int (*release) (struct gendisk *, fmode_t);
        int (*ioctl) (struct block_device *, fmode_t, unsigned, unsigned long);
        int (*compat_ioctl) (struct block_device *, fmode_t, unsigned, unsigned long);
        int (*direct_access) (struct block_device *, sector_t,
                                        void **, unsigned long *);
        unsigned int (*check_events) (struct gendisk *disk,
                              unsigned int clearing);
        /* ->media_changed() is DEPRECATED, use ->check_events() instead */
        int (*media_changed) (struct gendisk *);
        void (*unlock_native_capacity) (struct gendisk *);
        int (*revalidate_disk) (struct gendisk *);
        int (*getgeo)(struct block_device *, struct hd_geometry *);
        /* this callback is with swap_lock and sometimes page table lock held */
        void (*swap_slot_free_notify) (struct block_device *, unsigned long);
        struct module *owner;
};
```

A simple block device driver might only need to implement a subset of the methods defined for this structure, for example:

```
struct block_device_operations driver_blkdevops = {
```

The software described in this documentation is either in Extended Support or Sustaining Support. See https://www.oracle.com/us/support/library/enterprise-linux-support-policies-069172.pdf for more information.
Oracle recommends that you upgrade the software described by this documentation as soon as possible.


```
    .owner    = THIS_MODULE,
    .open     = driver_open,
    .release  = driver_release,
    .ioctl    = driver_ioctl,
    .getgeo   = driver_getgeo,
};
```

As well as the *driver_request()* and other driver methods, it is usual to define `module_init()` initialization and `module_exit()` cleanup routines for the driver that are called when a driver is loaded and unloaded. These routines should call `register_blkdev()` and `unregister_blkdev()` to register and unregister the device major number for the driver, and `alloc_gendisk()` and `del_gendisk()` to allocate and delete the kernel's representation of a generic disk (`gendisk`) device. To make the disk available to the system, an initialization routine usually calls `add_disk()` as its final action.

6.7 About Network Device Drivers

Network device drivers receive and transmit data packets on hardware interface that connect to external systems, and provide a uniform interface that network protocols can access. In Oracle Linux, these drivers are abstracted from the hardware implementation of the network adapters themselves, whose implementation hides the underlying layer-1 and layer-2 protocols.

Like a block driver's *driver_request()* routine, most callback routines of a network driver must be registered with the kernel to allow the kernel to execute them when data arrives on a network port or when the kernel needs to send data out on a network port. Like a character driver, a network driver does not access physically addressable storage media or support file system access. Unlike block and character device drivers, network drivers do not have device files under `/dev`. Instead, you use a `net_device` structure to define the capabilities of a network interface and the kernel updates the members of this structure when you use a command such as `ip` to configure the interface.

The implementation of network device drivers is likely to differ in many respects between Linux and UNIX operating systems, with most differences arising from the functions that the driver uses to handle network requests, the structures that are used to represent network driver methods and to define the properties of network interfaces, and the kernel interface routines for handling network queues.

The *driver_int()* and *driver_svr()* routines that UNIX network device drivers use to handle device interrupts and the transfer incoming data from a network device to the upstream protocol module are usually named *driver_interrupt()* and *driver_rx()* on Linux systems. The *driver_rx()* method should call `netif_rx()` to pass the socket buffer to the upstream module.

The *driver_put()* routine that UNIX network device drivers use to send data out on a network device corresponds to the `ndo_start_xmit()` method on Linux systems. The method can use the following utility functions to control upstream queueing of socket buffers for transmission:

`netif_queue_stopped()`
: Asks the kernel whether a queue is currently stopped.

`netif_start_queue()`
: Informs the kernel that the driver is ready to start sending packets.

`netif_stop_queue()`
: Asks the kernel to stop sending packets when transmission buffers are full or for driver cleanup when closing the device.

`netif_wake_queue()`
: Asks the kernel to wake up a queue that is currently stopped and resume sending packets.

After a network packet has been sent out successfully, *driver_interrupt()* calls a *driver_tx* routine that deletes the driver's copy of the socket buffer data.

The software described in this documentation is either in Extended Support or Sustaining Support. See https://www.oracle.com/us/support/library/enterprise-linux-support-policies-069172.pdf for more information.
Oracle recommends that you upgrade the software described by this documentation as soon as possible.

The net_device_ops structure for a network device, defined in <linux/netdevice.h>, contains a set of method pointers that specify how the system interacts with the device.

```
struct net_device_ops {
        int                     (*ndo_init)(struct net_device *dev);
        void                    (*ndo_uninit)(struct net_device *dev);
        int                     (*ndo_open)(struct net_device *dev);
        int                     (*ndo_stop)(struct net_device *dev);
        netdev_tx_t             (*ndo_start_xmit) (struct sk_buff *skb,
                                                   struct net_device *dev);
        u16                     (*ndo_select_queue)(struct net_device *dev,
                                                    struct sk_buff *skb);
        void                    (*ndo_change_rx_flags)(struct net_device *dev,
                                                       int flags);
        void                    (*ndo_set_rx_mode)(struct net_device *dev);
        void                    (*ndo_set_multicast_list)(struct net_device *dev);
        int                     (*ndo_set_mac_address)(struct net_device *dev,
                                                       void *addr);
        int                     (*ndo_validate_addr)(struct net_device *dev);
        int                     (*ndo_do_ioctl)(struct net_device *dev,
                                                struct ifreq *ifr, int cmd);
        int                     (*ndo_set_config)(struct net_device *dev,
                                                  struct ifmap *map);
        int                     (*ndo_change_mtu)(struct net_device *dev,
                                                  int new_mtu);
        int                     (*ndo_neigh_setup)(struct net_device *dev,
                                                   struct neigh_parms *);
        void                    (*ndo_tx_timeout) (struct net_device *dev);

        struct rtnl_link_stats64* (*ndo_get_stats64)(struct net_device *dev,
                                                     struct rtnl_link_stats64 *storage);
        struct net_device_stats* (*ndo_get_stats)(struct net_device *dev);

        void                    (*ndo_vlan_rx_register)(struct net_device *dev,
                                                        struct vlan_group *grp);
        void                    (*ndo_vlan_rx_add_vid)(struct net_device *dev,
                                                       unsigned short vid);
        void                    (*ndo_vlan_rx_kill_vid)(struct net_device *dev,
                                                        unsigned short vid);
        /* Several lines omitted */
};
```

A simple network device driver might only need to implement a subset of the functions defined for this structure, for example:

```
struct net_device_ops driver_netdevops = {
    .ndo_init       = driver_init,
    .ndo_open       = driver_open,
    .ndo_stop       = foo_close,
    .ndo_start_xmit = foo_start_xmit,
    .ndo_do_ioctl   = foo_ioctl,
    .ndo_tx_timeout = foo_tx_timeout,
};
```

The kernel calls ndo_open() when you bring up and assign an address to a network interface, ndo_stop() when you shut down the interface, and ndo_start_xmit() when it wants to transmit a packet.

The net_device structure for a character device, defined in <linux/netdevice.h>, defines the properties of the network interface.

The most important members of the net_device structure are:

The software described in this documentation is either in Extended Support or Sustaining Support. See https://www.oracle.com/us/support/library/enterprise-linux-support-policies-069172.pdf for more information.
Oracle recommends that you upgrade the software described by this documentation as soon as possible.

`dev_addr`
: The hardware media access control (MAC) address.

`features`
: A bit mask that describes the features that the device supports.

`flags`
: Describes the current properties of the device, such as the following:

`IFF_BROADCAST`	Broadcast mode is enabled.
`IFF_LOOPBACK`	The interface corresponds to a loopback device.
`IFF_MULTICAST`	Multicast mode is enabled.
`IFF_NOARP`	The interface does not use ARP to perform address resolution. This flag is usually set by point-to-point interfaces.
`IFF_PROMISC`	Promiscuous mode is enabled.
`IFF_UP`	The interface is enabled. The driver does not set this flag. Using an application such as `ip` to bring an interface up or down causes the kernel to invoke the `ndo_open()` or `ndo_stop()` method and set or unset the flag to indicate the state of the interface.

`irq`
: The interrupt request queue (IRQ) number.

`mtu`
: The maximum transmission unit (MTU), which is the maximum frame size that the device can transmit.

`name`
: The name of the interface, for example, `eth0` or `lo0`.

`netdev_ops`
: A pointer to a `net_device_ops` structure that defines the methods for the interface.

`promiscuity`
: A counter of how many client applications have placed the interface in promiscuous mode.

`stats`
: A `net_device_stats` structure that contains interface usage statistics.

As well as driver methods, *`driver_interrupt()`*, and related helper routines, it is usual to define `module_init()` initialization and `module_exit()` cleanup routines for the driver that are called when the driver is loaded and unloaded. These routines should call `register_netdev()` and `unregister_netdev()` to register and unregister the device, and `alloc_netdev()` and `free_netdev()` to allocate and delete the kernel's representation of the device. For Ethernet devices, it is usual to call `alloc_etherdev()` instead of `alloc_netdev()`. When initializing the driver, use `request_irq()` to install the interrupt handler, remembering to specify shared interrupts by setting the `SA_SHIRQ` bit in the *`flags`* argument as well as a unique *`dev_id`* argument that the handler can use to identify interrupts that it should process. The cleanup routine should call `free_irq()` to unregister the handler.

The software described in this documentation is either in Extended Support or Sustaining Support. See https://www.oracle.com/us/support/library/enterprise-linux-support-policies-069172.pdf for more information.
Oracle recommends that you upgrade the software described by this documentation as soon as possible.

6.8 About USB Device Drivers

As with other bus-based subsystems in Oracle Linux, a USB device driver registers its driver object with the USB subsystem. It can then use vendor and device identifiers to determine if the USB hardware is present.

The implementation of USB device drivers is likely to differ in many respects between Linux and UNIX operating systems, with most differences arising from the functions that the driver uses to handle USB requests, the structures that are used to represent USB driver methods and to define the properties of USB devices, and the kernel interface routines for communicating with the USB infrastructure.

The `usb_driver` and `usb_device_id` structures, defined in `<linux/usb.h>`, describe the USB driver and the USB device types that it supports. The `usb_driver` defines a number of methods that the USB core can use with the device:

```
struct usb_driver {
        const char *name;

        int (*probe) (struct usb_interface *intf,
                      const struct usb_device_id *id);

        void (*disconnect) (struct usb_interface *intf);

        int (*unlocked_ioctl) (struct usb_interface *intf, unsigned int code,
                       void *buf);

        int (*suspend) (struct usb_interface *intf, pm_message_t message);
        int (*resume) (struct usb_interface *intf);
        int (*reset_resume)(struct usb_interface *intf);

        int (*pre_reset)(struct usb_interface *intf);
        int (*post_reset)(struct usb_interface *intf);

        const struct usb_device_id *id_table;

        struct usb_dynids dynids;
        struct usbdrv_wrap drvwrap;
        unsigned int no_dynamic_id:1;
        unsigned int supports_autosuspend:1;
        unsigned int soft_unbind:1;
};
```

A driver should define `probe()` and `disconnect()` methods to support initialization and cleanup when a device is plugged in or removed from the system. The driver should call `usb_register()` and `usb_deregister()` to register the driver with the USB core and to unregister it.

The `urb` structure defines a USB request block that the kernel uses to send data to a USB device. To send data to a device, the driver's `write()` method should use `usb_alloc_urb()` to allocate an `urb`, `usb_buffer_alloc()` to allocate a buffer for the data, `copy_from_user()` to write the data into the buffer, `usb_fill_bulk_urb()` to fill in the details of the `urb`, and `usb_submit_urb()` to submit the urb to the USB core. The USB core invokes the driver's callback() method to report the status of the transaction. The driver should use `usb_buffer_free()` to free the allocated buffer.

To read data from a device, the driver's `read()` method can use `usb_bulk_msg()`.

The driver should use `usb_control_msg()` to send USB control messages.

Other useful helper functions include `usb_get_descriptor()` and `usb_get_string()` to retrieve device descriptor information or a string about a USB device.

The software described in this documentation is either in Extended Support or Sustaining Support. See https://www.oracle.com/us/support/library/enterprise-linux-support-policies-069172.pdf for more information.
Oracle recommends that you upgrade the software described by this documentation as soon as possible.

6.9 About the Sysfs File System

The `sysfs` virtual file system presents a user-spec view of the devices in the system, which changes dynamically whenever a device is inserted or removed. The directory hierarchy under `/sys` represents the global, bus, and device discovery layers. The files in the global layer report the power state of the device, which can also be set. The files in the bus layer report information such as the IRQ number and resources for each device. A device driver can also expose data or tunable interfaces via files in the file system.

6.10 Loading Device Drivers as Kernel Modules

Platform-specific differences are likely in the system administration commands and system calls that are used to administer device drivers.

Oracle Linux treats kernel modules and device drivers in the same way. Both kernel modules and device drivers can be added by using `modprobe` command, which is also used by the kernel module daemon `kmod`.

`modprobe` consults the module dependencies file `/lib/modules/version/modules.dep` to see if it must load any other modules before the requested module, and then runs `insmod` to load the prerequisite modules.

`insmod` uses the query_module() system call to retrieve the symbol table for a module and the create_module() system call to set up a module entry and reserve memory for the module before linking the module into the kernel and calling the `init_module()` system call to initialize the module.

If you use `modprobe` to delete a module, the command invokes the `delete_module()` system call.

Typically, kernel module compilation on Oracle Linux looks similar to the following command:

```
$ gcc -D__KERNEL__ -O2 -DMODULE -W -Wall -Wstrict-prototypes -Wmissing-prototypes \
  -isystem /lib/modules/`uname -r`/build/include -c flkm.c -o flkm
```

The software described in this documentation is either in Extended Support or Sustaining Support. See https://www.oracle.com/us/support/library/enterprise-linux-support-policies-069172.pdf for more information. Oracle recommends that you upgrade the software described by this documentation as soon as possible.

Chapter 7 Security

Table of Contents

7.1 Physical Security 43
7.2 Delegate Minimal Privileges as Appropriate 43
7.3 About Discretionary and Mandatory Access Control Policies 44
7.4 About Targeted and Multilevel Security Policies 44
7.5 About Security Contexts and Users 45
7.6 Ensure Strong Defenses 45
7.7 Encryption Algorithms, Mechanisms, and Mapping 46

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. Oracle Linux provides enterprise-class features that you can depend on to protect your applications by combining multiple security technologies to process and user rights as well as unmatched monitoring and auditing capabilities.

Regardless of the hosting operating system, security administrators follow simple rules to build a secure system:

- Ensure physical security
- Deploy stringent access controls
- Simplify administration
- Delegate appropriate (minimal) privileges
- Do minimal installs
- Ensure strong defenses

This chapter discusses some of these aspects of ensuring system security in more detail.

7.1 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.

7.2 Delegate Minimal Privileges 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. Oracle Linux, like traditional UNIX systems, follows a superuser-based model. Applications check the ID of the user (such as 0 for

The software described in this documentation is either in Extended Support or Sustaining Support. See https://www.oracle.com/us/support/library/enterprise-linux-support-policies-069172.pdf for more information. Oracle recommends that you upgrade the software described by this documentation as soon as possible.

`root`) to test for the availability of specific privileges. The `sudo` command allows a user to execute a command as `root` or another specified user, provided that they have been granted permission in the `/etc/sudoers` file. If you want to grant certain users authority to be able to perform specific administrative tasks via `sudo`, you can use the `visudo` command to modify the contents of this file.

By default, an Oracle Linux system is configured so that you cannot log in directly as `root`. You must log in as a named user before using either `su` or `sudo` to perform tasks as `root`. This configuration allows system accounting to trace the original login name of any user who performs a privileged administrative action.

You can also configure SELinux to provide Role-Based Access Control (RBAC). Under this security model, a user's membership of an SELinux domain determines which processes and files he or she can run or access on the system.

7.3 About Discretionary and Mandatory Access Control Policies

Traditional Linux security is based on a Discretionary Access Control (DAC) policy, which provides minimal protection from broken software or from malware that is running as a normal user or as `root`. Access to files and devices is based solely on user identity and ownership. Malware or broken software can do anything with files and resources that the user that started the process can do. If the user is `root` or the application is `setuid` or `setgid` to `root`, the process can have `root`-access control over the entire file system.

The National Security Agency created Security Enhanced Linux (SELinux) to provide a finer-grained level of control over files, processes, users and applications in the Linux operating system. The SELinux enhancement to the Linux kernel implements the Mandatory Access Control (MAC) policy, which allows you to define a security policy that provides granular permissions for all users, programs, processes, files, and devices. The kernel's access control decisions are based on all the security relevant information available, and not solely on the authenticated user identity.

The features of SELinux enable an organization to define and implement a security policy on an Oracle Linux system. A security policy is the set of rules and practices that help protect information and other resources, such as computer hardware, at your site. Typically, security rules handle such issues as who has access to which information or who is allowed to write data to removable media. Security practices are recommended procedures for performing tasks.

When security-relevant access occurs, such as when a process attempts to open a file, SELinux intercepts the operation in the kernel. If a MAC policy rule allows the operation, it continues; otherwise, SELinux blocks the operation and returns an error to the process. The kernel checks and enforces DAC policy rules before MAC rules, so it does not check SELinux policy rules if DAC rules have already denied access to a resource.

SELinux should normally run in Enforcing mode where the kernel denies access to users and programs unless permitted by SELinux security policy rules. All denial messages are logged as AVC (Access Vector Cache) denials. If SELinux runs in Permissive mode, the kernel does not enforce security policy rules but SELinux sends denial messages to a log file. This allows you to see what actions would have been denied if SELinux were running in enforcing mode. This mode is intended to used for diagnosing the behavior of SELinux. If you disable SELinux, the kernel uses only DAC rules for access control.

7.4 About Targeted and Multilevel Security Policies

An SELinux policy describes the access permissions for all users, programs, processes, and files, and for the devices upon which they act. You can configure SELinux to implement either Targeted Policy or Multilevel Security (MLS) Policy.

The software described in this documentation is either in Extended Support or Sustaining Support. See https://www.oracle.com/us/support/library/enterprise-linux-support-policies-069172.pdf for more information.
Oracle recommends that you upgrade the software described by this documentation as soon as possible.

Targeted Policy applies access controls to a limited number of processes that are believed to be most likely to be the targets of an attack on the system. Targeted processes run in their own SELinux domain, known as a *confined domain*, which restricts access to files that an attacker could exploit. If SELinux detects that a targeted process is trying to access resources outside the confined domain, it denies access to those resources and logs the denial. Only specific services run in confined domains. Examples are services that listen on a network for client requests, such as `httpd`, `named`, and `sshd`, and processes that run as `root` to perform tasks on behalf of users, such as `passwd`. Other processes, including most user processes, run in an unconfined domain where only DAC rules apply. If an attack compromises an unconfined process, SELinux does not prevent access to system resources and data.

Multilevel Security (MLS) Policy applies access controls to multiple levels of processes with each level having different rules for user access. Users cannot obtain access to information if they do not have the correct authorization to run a process at a specific level. In SELinux, MLS implements the Bell–LaPadula (BLP) model for system security, which applies labels to files, processes and other system objects to control the flow of information between security levels. In a typical implementation, the labels for security levels might range from the most secure, `top secret`, through `secret`, and `classified`, to the least secure, `unclassified`. For example, under MLS, you might configure a program labelled `secret` to be able to write to a file that is labelled `top secret`, but not to be able to read from it. Similarly, you would permit the same program to read from and write to a file labelled `secret`, but only to read `classified` or `unclassified` files. As a result, information that passes through the program can flow upwards through the hierarchy of security levels, but not downwards.

7.5 About Security Contexts and Users

Under SELinux, all file systems, files, directories, devices, and processes have an associated security context. For files, SELinux stores a context label in the extended attributes of the file system. The context contains additional information about a system object: the SELinux user, their role, their type, and the security level. SELinux uses this context information to control access by processes, Linux users, and files.

An SELinux user account compliments each regular Oracle Linux user account. SELinux maps every Oracle Linux user to an SELinux user identity that is used in the SELinux context for the processes in a user session.

In the Role-Based Access Control (RBAC) security model, a role acts as an intermediary abstraction layer between SELinux process domains or file types and an SELinux user. Processes run in specific SELinux domains, and file system objects are assigned SELinux file types. SELinux users are authorized to perform specified roles, and roles are authorized for specified SELinux domains and file types. A user's role determines which process domains and file types he or she can access, and hence, which processes and files, he or she can access.

SELinux users form part of a SELinux policy that is authorized for a specific set of roles and for a specific MLS (Multi-Level Security) range, and each Oracle Linux user is mapped to an SELinux user as part of the policy. As a result, Linux users inherit the restrictions and security rules and mechanisms placed on SELinux users. To define the roles and levels of users, the mapped SELinux user identity is used in the SELinux context for processes in a session.

7.6 Ensure Strong Defenses

The Internet comprises hundreds of thousands of networks that are interconnected without boundaries. In today's business environment, you will rarely find any standalone servers that cater for 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

The software described in this documentation is either in Extended Support or Sustaining Support. See https://www.oracle.com/us/support/library/enterprise-linux-support-policies-069172.pdf for more information.
Oracle recommends that you upgrade the software described by this documentation as soon as possible.

across the world and is, therefore, potentially vulnerable to threats from individuals who might not have any physical access. Oracle Linux 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.

Oracle Linux provides standard industry interfaces for network security such as PAM, GSS-API, SASL, and PKCS#11, eliminating any need for you to write, maintain, and optimize cryptographic algorithms. Oracle Linux provides optimized cryptographic mechanisms as part of the operating system. This cryptographic framework is the backbone of cryptographic services in Oracle Linux and provides standard PKCS #11 interfaces to accommodate consumers and providers of cryptographic services.

Consumers of cryptographic services need not have any specific knowledge of the installed cryptographic mechanisms. Similarly, providers of cryptographic services can support different types of consumers. You do not have to write consumer-specific code in the providers.

7.7 Encryption Algorithms, Mechanisms, and Mapping

Oracle Linux 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 seamless manner. Applications can also make use of direct cryptographic services, such as encryption and decryption of files. If a cryptographic accelerator is integrated with the CPU, the cryptographic framework can take advantage of this hardware on behalf of applications.

The available function calls are described in the following manual pages:

- Arcfour (RC4) in `rc4(3)`
- Blowfish in `blowfish(3)`
- DES in `des(3)`
- MD4 and MD5 in `md5(3)`
- RSA in `RSA_set_method(3)`
- SHA1 in `sha(3)`

For information about hardware-assisted encryption and how to benefit from it without making fundamental changes to application code, see *Oracle Advanced Security Transparent Data Encryption Best Practices*.

For more information about configuring system security, see *Oracle® Linux 6: Security Guide*.

The software described in this documentation is either in Extended Support or Sustaining Support. See https://www.oracle.com/us/support/library/enterprise-linux-support-policies-069172.pdf for more information.
Oracle recommends that you upgrade the software described by this documentation as soon as possible.

Chapter 8 Runtime Environment

Table of Contents

8.1 Runtime Limits 47
8.2 Migrating Scripts 47
8.3 Managing Services 47

Although Oracle Linux is similar to a UNIX-based operating system, there are some fundamental differences in its runtime environment. This chapter discusses some differences that you might encounter.

8.1 Runtime Limits

When moving to Oracle Linux, one reason an application might face problems are any differences in the resource limits on the two platforms such as the settings for the interprocess communication facilities (shared memory, semaphores, and messages).

On Oracle Linux, the `ipcs -l` command displays output similar to the following for the interprocess communication parameter settings (the added `#` comments show the equivalent parameter setting in `/etc/sysctl.conf`).

```
# ipcs -l

------ Shared Memory Limits --------
max number of segments = 4096                # kernel.shmmni
max seg size (kbytes) = 67108864             # kernel.shmmax (units of bytes)
max total shared memory (kbytes) = 17179869184 # kernel.shmall (units of 4KB pages)
min seg size (bytes) = 1                     # no available parameter

------ Semaphore Limits --------
max number of arrays = 128                   # kernel.sem (argument 4)
max semaphores per array = 250               # kernel.sem (argument 1)
max semaphores system wide = 32000           # kernel.sem (argument 2)
max ops per semop call = 32                  # kernel.sem (argument 3)
semaphore max value = 32767                  # no available parameter

------ Messages: Limits --------
max queues system wide = 7917                # kernel.msgmni
max size of message (bytes) = 65536          # kernel.msgmax
default max size of queue (bytes) = 65536    # kernel.msgmnb
```

If you modify the parameters in `/etc/sysctl.conf`, note that the units used in that file do not always match those that `ipcs` displays. To load the new settings from this file, use the `sysctl -p` command.

8.2 Migrating Scripts

UNIX applications and development environments commonly use shell scripts, Perl scripts, and scripts written in other languages. If the scripting language used on the source platform is also supported in Oracle Linux, migrating the scripts is a straightforward exercise. When migrating legacy scripts, the availability of the command and any command-line options on both the source and target platforms is critical in determining the migration effort.

8.3 Managing Services

A UNIX daemon is a program that runs in the background and is independent of control from a terminal. Daemons are usually started by a system startup script, where there is no controlling terminal although

The software described in this documentation is either in Extended Support or Sustaining Support. See https://www.oracle.com/us/support/library/enterprise-linux-support-policies-069172.pdf for more information.
Oracle recommends that you upgrade the software described by this documentation as soon as possible.

they can also be started from the command line if required. The `init` daemon is the system and service manager for Oracle Linux. It is one of the first processes that starts at boot time with a PID of 1 and is the ancestor of all processes. Services are started and stopped through `init` scripts in the `/etc/init.d` directory. Most 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 run level *N*. As this model involved having multiple copies of the same script in many different directories,

Oracle Linux adopts the standard of putting all service control scripts in the `/etc/init.d/` directory and symbolically linking these scripts from the `/etc/rcN.d/` directories. With this arrangement, it is possible to use centralized commands such as `chkconfig` and `service` to manage all services from a single interface.

The `service` command 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, reload, and perform other actions on services. As 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 by specifying the `--status-all` option:

```
# service --status-all
abrtd (pid  2031) is running...
abrt-dump-oops (pid 2039) is running...
acpid (pid  1669) is running...
atd (pid  2146) is running...
auditd (pid  1407) is running...
automount (pid  1817) is running...
...
```

You can use the `chkconfig` command to query and modify the system run level at which a service starts. For example, to display the current settings for the `httpd` service:

```
# chkconfig --list httpd
httpd           0:off 1:off 2:on 3:on 4:on 5:on 6:off
```

The output shows that `crond` starts automatically at boot time for run levels 2, 3, 4, and 5.

You can use `chkconfig` to prevent a service from startign at certain run levels, for example:

```
# chkconfig --level 34 httpd off
# chkconfig --list httpd
httpd           0:off 1:off 2:on 3:off 4:off 5:on 6:off
```

You can also use `chkconfig` to disable a service altogether, for example:

```
# chkconfig httpd off
# chkconfig --list httpd
httpd           0:off 1:off 2:off 3:off 4:off 5:off 6:off
```

The `chkconfig` command does not affect the state of the service until the run level changes. To disable or enable a service immediately, use the `service` command, for example:

```
# service httpd stop
Stopping httpd:                                            [  OK  ]
# service httpd start
Starting httpd:                                            [  OK  ]
```

The software described in this documentation is either in Extended Support or Sustaining Support. See https://www.oracle.com/us/support/library/enterprise-linux-support-policies-069172.pdf for more information.
Oracle recommends that you upgrade the software described by this documentation as soon as possible.

Chapter 9 Pluggable Authentication Modules

Table of Contents

9.1 About Pluggable Authentication Module (PAM) 49
9.2 About PAM Operation for an Application 50
9.3 PAM Implementation Differences 51

Almost every enterprise application requires services such as authentication, logging, persistence, and security. In most applications, each service is either developed by the application developer, is reused after customizing the offerings provided by different vendors, or is implemented by leveraging the frameworks provided by the operating system.

Applications developed using a framework are interoperable with open standards. Usually, 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. A framework allows you to develop a structured, compliant application that is portable, maintainable, and upgradable with changing business rules and compliance requirements.

If your application uses a framework provided by the operating system, for example, a security framework,instead of having a custom-built implementation, a file-system framework, a cryptographic framework, or a hot-plug framework, migration from one platform to another becomes simple. Most frameworks available on UNIX systems are available on Oracle Linux and maintain similar, if not identical, 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). It describes differences in implementation and points that you should consider when migrating an application to Oracle Linux.

9.1 About 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 is the central element in the PAM architecture. It exports an API (see the `pam(3)` manual page) that applications can call for authentication, account management, credential establishment, session management, and password changes. The `libpam` library imports configuration files, either separate files under `/etc/pam.d` or the `/etc/pam.conf` configuration file, that specify the PAM module requirements for each available service.

Oracle Linux provides a PAM infrastructure that is similar to that on other platforms. Although the functionality might be similar, there could be subtle differences between the implementations. For example, PAM configuration is usually set by editing individual configuration files located in the `/etc/pam.d` directory. The presence of this directory causes PAM to ignore the legacy PAM configuration file `/etc/pam.conf`.

The software described in this documentation is either in Extended Support or Sustaining Support. See https://www.oracle.com/us/support/library/enterprise-linux-support-policies-069172.pdf for more information.
Oracle recommends that you upgrade the software described by this documentation as soon as possible.

PAM on Oracle Linux does not support the control value `binding` that you might find on other operating systems. When `binding` is specified, if the service module returns success and no preceding required modules return failures, PAM immediately returns success without calling any subsequent modules. If a module returns failure, PAM treat the failure as a required module failure, and continues to process the PAM stack.

9.2 About PAM Operation for an Application

An application typically performs the following steps during the invocation of a typical PAM session.

1. Call `pam_start()` 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);
   ```

   `pam_start()` returns a PAM session handle *`pamh`* for use with subsequent function calls.

2. Obtain information relating to the transaction (such as the applicant's user name and the name of the host on which the client runs) and use `pam_set_item()` to submits it to PAM.

   ```
   int pam_set_item(pam_handle_t *pamh, int item_type, const void *item);
   ```

3. Call `pam_authenticate()` to authenticate the applicant.

   ```
   int pam_authenticate(pam_handle_t *pamh, int flags);
   ```

4. Once the user has been authenticated, call `pam_acct_mgmt()` to establish whether the account is valid and the user is permitted to log in at this time. Optionally, modules of type account-management 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.

   ```
   int pam_acct_mgmt(pam_handle_t *pamh, int flags);
   ```

   If the password is correct but has expired, `pam_acct_mgmt()` returns `PAM_NEW_AUTHTOK_REQD` instead of `PAM_SUCCESS`.

   If the function returns `PAM_NEW_AUTHTOK_REQD`, call `pam_chauthtok()` to force the client to change the authentication token for the requested account.

   ```
   int pam_chauthtok(pam_handle_t *pamh, const int flags);
   ```

5. Call `pam_setcred()` to establish the identity of the user, which can include credentials such as access tickets and supplementary group memberships.

   ```
   int pam_setcred(pam_handle_t *pamh, int flags);
   ```

6. When the credentials have been established, call `pam_open_session()` to open and configure the session, which typically includes performing tasks such as making system resources available (for example, mounting a user's home directory) and establishing an audit trail.

   ```
   int pam_open_session(pam_handle_t *pamh, int flags);
   ```

7. To close the session, call `pam_close_session()`.

The software described in this documentation is either in Extended Support or Sustaining Support. See https://www.oracle.com/us/support/library/enterprise-linux-support-policies-069172.pdf for more information.
Oracle recommends that you upgrade the software described by this documentation as soon as possible.


```
int pam_close_session(pam_handle_t *pamh, int flags);
```

8. Call `pam_end()` to notify the PAM library that the application has finished processing a transaction and that it can release whatever resources it has allocated in the course of the transaction.

```
int pam_end(pam_handle_t *pamh, int status);
```

9.3 PAM Implementation Differences

The data structures used by PAM might differ between platforms. For example, the `pam_message` structure is defined as follows:

```
struct pam_message {
    int msg_style;
    const char *msg;
};
```

On Oracle Linux, PAM interprets the `msg` argument as equivalent to the following prototype:

```
const struct pam_message *msg[]
```

Functions such as `pam_set_item()` and `pam_get_item()` are defined in `<security/_pam_types.h>` which is included by `<security/pam_modules.h>`.

The return values from PAM functions differ between platform implementations. For example, a function might return `PAM_BAD_ITEM` instead of `PAM_OPEN_ERR` if an error occurs when opening a service module.

The software described in this documentation is either in Extended Support or Sustaining Support. See https://www.oracle.com/us/support/library/enterprise-linux-support-policies-069172.pdf for more information.
Oracle recommends that you upgrade the software described by this documentation as soon as possible.

Chapter 10 Packaging and Distributing Applications

Table of Contents

10.1 About RPM Packaging 53
    10.1.1 About RPM Categories 54
    10.1.2 Administering RPMs 54
    10.1.3 Building an RPM 55
10.2 About Oracle Enterprise Manager 56
10.3 About Spacewalk 57

With the increasing trend toward consolidation by the use of virtualization, system administrators have to manage large volumes of software, often with complex interdependencies, on a wide range of hardware platforms. Keeping such systems, with complex software dependencies on varied hardware platforms, running smoothly is a complex and challenging task. The proper management of system software can help ensure a well-known, tested, and consistent system state across the systems in the data center. For most enterprise-level IT organizations, a significant effort is required to upgrade the operating system software to the appropriate patch levels for bug fixes, security updates, and new hardware driver support.

While a simple application typically consist of only a few executable files, most enterprise software applications are more complex, consisting of many executable files, libraries, scripts, configuration files, and documentation notes and guides. All these files are delivered via a package that includes information about where to place the files in the file system and the permissions they should be assigned. There are many choices of packaging format and some are easier to use than others. The packaging mechanism that you should use to deliver an application depends on the requirements of the application as well as upgrade and maintainability considerations. Simple packaging requirements can be met by using a simple packaging mechanism such as a tarball, or you might need to use a complex and advanced packaging mechanism such as RPM.

This chapter provides an overview of how to administer and create RPMs on Oracle Linux. It also provides links to more information about the Oracle Enterprise Manager and Spacewalk IT management products that you can use to create and deploy software packages for your systems.

10.1 About RPM Packaging

On Oracle Linux, the RPM package manager 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, consider the package file that contains the GNU Compiler Collection is named `gcc-4.4.6-4.el6.x86_64.rpm`, where the package version is 4.4.6, the build number is 4, and the supported architecture and operating system are x86-64 and Oracle Linux 6.

You can use the RPM manager to copy an `rpm` file to another Linux system and install it, display the complete contents of the package, or remove or update the package.

An RPM file usually contains four sections:

The software described in this documentation is either in Extended Support or Sustaining Support. See https://www.oracle.com/us/support/library/enterprise-linux-support-policies-069172.pdf for more information.
Oracle recommends that you upgrade the software described by this documentation as soon as possible.

- An identification area, which contains information about the RPM package including the version of the RPM packing system.
- A signature, which contains size, checksums, and other information that can be used to check the authenticity and integrity of a package.
- A preamble, which contains tagged data that provides information about the contents of the package. The tags in this section contains mandatory and optional information. For example, the mandatory `NAME` tag defines the package name, and the optional `PRE` tag defines a pre-installation script that the rpm command runs before installing the files. The tags have the format: *`tag:data`*. Types of tag that are typically found in an RPM package include:
  - Architecture-specific tags
  - Dependency tags
  - Descriptive tags
  - Directory-related tags
  - Operating system tag
  - Package naming tags
  - Source and patch tags

  For example, the Dependency tag category includes tags such as `requires`, `conflicts`, and `provides`.
- The payload section specifies the files to be installed on the target system.

10.1.1 About RPM Categories

RPMs can contain either binary or source packages.

A binary RPM is compiled for a particular target architecture such as `x86_64`, or `noarch` if its contents do not depend on the architecture as is the case for documentation and scripts. An example of a binary RPM might contain the GCC compiler and tools compiled for an Intel x86-64 target. Separate packages are required for each hardware platform that a package supports.

A source RPM provides the source code for binary packages and allows you to create a binary RPM.

10.1.2 Administering RPMs

In Oracle Linux, you can use the `yum` utility to install or upgrade RPM packages. The main benefit of using yum is that it also installs or upgrades any package dependencies. `yum` downloads the packages from repositories such as those that are available on the Unbreakable Linux Network (ULN) or the Oracle Linux yum server, but you can also set up your own repositories for use by systems that do not have Internet access. This section describes how to use the `rpm` command to administer packages that are not available in a yum repository.

Note

You need to be `root` to install, upgrade or erase an RPM.

The software described in this documentation is either in Extended Support or Sustaining Support. See https://www.oracle.com/us/support/library/enterprise-linux-support-policies-069172.pdf for more information.
Oracle recommends that you upgrade the software described by this documentation as soon as possible.

To install or upgrade a package using `rpm`, specify the `-U` option:

```
# rpm -U package.rpm
```

If you specify the `--test` option, `rpm` tests the installation or upgrade process, but it does not install the package:

```
# rpm -U --test package.rpm
```

To remove a package (*erase* in RPM terminology), use the `-e` option:

```
# rpm -e package
```

To list every RPM package installed on a system:

```
# rpm -qa
```

You can filter the output using `grep` and `sort` if you are searching for a particular package, for example:

```
# rpm -qa | grep gcc | sort
gcc-4.4.6-4.el6.x86_64
gcc-c++-4.4.6-4.el6.x86_64
gcc-gfortran-4.4.6-4.el6.x86_64
libgcc-4.4.6-4.el6.i686
libgcc-4.4.6-4.el6.x86_64
```

To verify that an individual package has been installed, specify the package name, which can be in short form:

```
# rpm -q package
```

For example:

```
# rpm -q gcc-c++
gcc-c++-4.4.6-4.el6.x86_64
```

To display information about a package, use the `-i` option:

```
# rpm -qi package
```

To list all the files in a package, use the `-l` option:

```
# rpm -ql package
```

For more information about using `rpm`, see the `rpm(8)` manual page.

10.1.3 Building an RPM

Note

You do not need to be `root` to build an RPM.

To build an RPM:

1. Create a tarball of the source code files for the package, for example:

   ```
   $ tar zcvf mypkg-1.0.tar.gz mypkg-1.0
   ```

2. Copy the tarball to the `SOURCES` directory under `/root/rpmbuild`.

The software described in this documentation is either in Extended Support or Sustaining Support. See https://www.oracle.com/us/support/library/enterprise-linux-support-policies-069172.pdf for more information.
Oracle recommends that you upgrade the software described by this documentation as soon as possible.


```
$ cp mypkg-1.0.tar.gz /root/rpmbuild/SOURCES
```

If you configure `SOURCES` in a different directory, you can use the `--root` option to specify this directory to the `rpmbuild` command.

3. Create a specification (`spec`) file, which contains information about your package such as its name, the version, the release number, which packages are also required, and the name of the tarball. The file should contain at least the following sections:

`%prep`	Contains the commands to prepare for the build.
`%build`	Contains the commands to build the software. Usually, only `make` is required as most of the required instructions appear in the makefile.
`%install`	Contains the commands to install the newly built application or library.
`%clean`	Contains the commands to clean up any files that the commands in the previous sections create.
`%files`	Lists the files that go into the binary RPM along with their file attributes.

See the `spec` files under `/root/rpmbuild/SPECS` for examples of how to create a `spec` file for your package.

You can use the `find-requires` and `find-provides` scripts in `/usr/lib/rpm` to determine Perl, Python, Tcl script, Java package, and other dependencies. `find-requires` determines the shared libraries that are required by the files specified on the standard input. `find-provides` determines the shared libraries that a package provides from the files specified on the standard input.

4. To build both a source RPM and a binary RPM, specify the `-ba` option and the name of the specification file to the `rpmbuild` command:

```
$ rpmbuild -ba spec_file
```

To build only the binary RPM, specify the `-bb` option instead.

Having built the packages, you can publish them on a Web server for downloading.

For more information about creating RPM packages, see the `rpmbuild(8)` manual page and the RPM documentation at http://www.rpm.org/wiki/Docs.

10.2 About Oracle Enterprise Manager

The Oracle Enterprise Manager family of products provide a complete, integrated enterprise cloud management solution for deploying, operating, testing and monitoring systems and for diagnosing and resolving problems in complex IT environments. By leveraging the built-in management capabilities of the Oracle stack for traditional and cloud environments, Oracle Enterprise Manager allows you to achieve unprecedented efficiency gains and dramatically increase service levels.

The key capabilities of Enterprise Manager include:

- A complete cloud life-cycle management solution that allows you to quickly set up, manage and support enterprise clouds, and traditional Oracle IT environments from applications to disk.

The software described in this documentation is either in Extended Support or Sustaining Support. See https://www.oracle.com/us/support/library/enterprise-linux-support-policies-069172.pdf for more information.
Oracle recommends that you upgrade the software described by this documentation as soon as possible.

- Maximum return on IT management investment by providing the best solutions for intelligent management of the Oracle stack and engineered systems through real-time integration of Oracle's knowledge base with your IT environment.
- Best service levels for traditional and cloud applications through business-driven application management.

For more information, see https://www.oracle.com/technetwork/oem/enterprise-manager/overview/index.html.

10.3 About Spacewalk

Spacewalk is an open source solution for systems management that allows you to:

- Create inventories of hardware and software.
- Provision, configure, monitor, and control systems.
- Create customized software packages.
- Manage software installations and updates.

If you have an Oracle Linux Basic or Premier Support subscription, you can access a fully supported build of Spacewalk 2.2 for managing Oracle Linux systems. This implementation of Spacewalk is provided as a transitional solution that you can use while you are planning a migration to Oracle Enterprise Manager from management tools such as Red Hat Satellite Server.

For more information, see *Oracle® Linux Manager & Spacewalk for Oracle® Linux Documentation*.