ORACLE

SOLARIS

An Oracle White Paper
May 2012

Oracle Solaris 11 ISV Adoption Guide

ORACLE

Executive Overview..... 1
Introduction..... 2
Recommended Strategy..... 3
Oracle Solaris Binary Guarantee..... 3
Oracle Solaris Zone Support..... 4

- Features with High Impact to ISVs..... 6
- Features with Medium Impact to ISVs..... 7
- Features with Limited or No Impact on ISVs..... 8
- Creating and installing an Oracle Solaris 10 Zone on Oracle Solaris 11 based on an Existing Oracle Solaris 10 System or Flash Archive Image..... 10
- Creating an Oracle Solaris 10 Zone in Oracle Solaris 11 based on a Single Oracle Solaris 10 Native Zone..... 12
- Creating an Oracle Solaris 10 Zone on Oracle Solaris 11 based on an System with Oracle Solaris 10 Native Zone (P2V and V2V)..... 12
- Compatibility Issues That Could Prevent an Application from Running in a Oracle Solaris Zone..... 12
- Reduced Process Privileges..... 13
- Libraries..... 13
- Networking..... 14
- Devices..... 14
- Networking..... 15
- Root File System..... 17
- Other Restrictions..... 17
- Command Line Modernization..... 18
- The Default Path..... 19

Oracle Solaris 11 Packaging Mechanisms..... 22

- IPS Packages..... 23
- Installation of Packages..... 24

POSIX compliant Header Files..... 26
Selected Removed Packages and Frameworks..... 26
Internationalization..... 29

- Removed @euro locales for territories: ES, DK, AT, DE, GR, IE, FI, FR, IT, NL, PT..... 29
- Removal of Short Form Locales..... 29

Tools to Manage the Transition to Oracle Solaris 11 31

- Introduction..... 31
- Compiler Suite..... 31
- Selecting Appropriate Compiler Flags..... 32
- 64-bit or 32-bit code..... 33
- Advanced compiler optimizations.....33
- Using the C++ Standard Library..... 34
- Linking Applications..... 34
- Writing portable applications..... 35
- Writing Parallel Applications..... 35
- Analysis Suite..... 36
- The Performance Analyzer..... 36
- The Code Analyzer..... 36
- The Thread Analyzer..... 37
- Oracle Solaris Studio IDE..... 37
- Introduction..... 38
- Binary / ELF Analyzer:..... 38
- Source Code Analyzer:..... 39
- RunTime Analyzer:..... 39
- Obtaining the Oracle Solaris 11 Preflight Application Checker 40

Conclusion..... 40

Executive Overview

This document is designed to help independent software vendors (ISVs) transition their applications smoothly from Oracle Solaris 10 to Oracle Solaris 11. It provides a list of all currently known modifications, which may be needed for existing applications to run on Oracle Solaris 11. The majority of applications will work without any modification. However, applications that rely on outdated frameworks and undocumented APIs may require minor changes.

Introduction

Oracle Solaris 11 is a major Oracle Solaris release that reconciles all technological innovations published in recent years through OpenSolaris and Oracle Solaris 11 Express with features known and used in Oracle Solaris 10. Most applications are known to work on Oracle Solaris 11 without changes. In addition, Oracle Solaris 11 provides two major options that allow the vast majority of applications to run Oracle Solaris:

- Most Oracle Solaris 10 applications will run "as is" on Oracle Solaris 11
- Those applications that rely on Oracle Solaris 10 features that are not directly supported in Oracle Solaris 11 will run in an Oracle Solaris 10 virtual environment (Oracle Solaris 10 Zones)

Because of this two-pronged approach, strict compatibility issues should be less problematic with Oracle Solaris 11 compared to previous major Oracle Solaris releases. This document will put a strong emphasis on the pragmatic aspect of "making the application work". This document represents a snapshot in time. It discusses the currently known border cases where configuration changes may be needed due to:

- Modified default configurations and settings (examples: default shells, ZFS root file system)
- Retired packages (example: obsolete physical devices)
- Evolution of industry standards (example: standardized locale names)
- Removal of already-deprecated functions
- Removal of undocumented and unsupported functionality
- Border cases introduced by new technologies (such as ZFS root file system default)
- Bug fixes and increased security (narrower access and usage rights)
- The new installation technology known as Image Packaging System (IPS)

Disclaimer: This document will be updated with any subsequent releases of Oracle Solaris 11 general availability.

Recommended Strategy

Release managers will want to:

- Use this document primarily as an overall "checklist" and consult the Oracle Solaris product documentation itself for detailed information.
- Use the Oracle Solaris 11 compatibility checking tool, which will check Oracle Solaris 10 applications for their readiness with Oracle Solaris 11..
- Consult the EOF (End Of Feature) section in the latest Oracle Solaris 10 and Oracle Solaris 11 release notes. This section lists all functionality, which is planned to be removed at a later point of time.
- Consult the Oracle Solaris 11 – End of Feature Notices web page[1]. This is a new service that documents frame works and features that will be removed in Oracle Solaris 11..

Oracle Solaris Binary Guarantee

Oracle Solaris 11 Express and Oracle Solaris 11 are binary compatible with previous Oracle Solaris versions[2]. This means that existing application binaries run on Oracle Solaris 11 without recompilation, as long as that applications were coded to standard, published Oracle Solaris APIs and ABIs (see below).

Oracle Solaris Application Binary Interface

An ABI[3] (Application Binary Interface) defines the runtime interface between an application and the operating system or other applications. In addition to defining a set of build-time APIs (Application Programming Interfaces), it also defines the size of fundamental data types, such as `int` and `float`, layout and alignment of data structures, organization of the stack, calling conventions, register allocation, system call mechanisms and executable file formats.
Oracle Solaris defines four ABIs, the 32-bit and 64-bit versions for the SPARC and x86/x64 architectures. Each Oracle Solaris ABI consists of a set of supported run-time interfaces that are available for an application to use with the Oracle Solaris Operating system, consisting of:

- The Oracle Solaris system library APIs defined in section 3 of the Unix manual
- The system calls provided by the Oracle Solaris kernel as defined in section 2 of Unix manual
- The locations and formats of various system files, which are documented in section 4 of the Unix manual

[1] http://www.oracle.com/technetwork/systems/end-of-notices/eonsolaris11-392732.html

[2] http://www.oracle.com/us/products/servers-storage/solaris/solaris-guarantee-program-1426902.pdf

[3] http://download.oracle.com/docs/cd/E19253-01/817-4415/817-4415.pdf

- The input and output syntax and semantics of Oracle Solaris utilities, which are documented in section 1 of the Unix manual

For application developers, the C language APIs defined by the Oracle Solaris system libraries are the most important part of the Oracle Solaris ABI. Applications written in other languages must use these C API's to interface with the Oracle Solaris kernel.

Oracle Solaris provides a binary guarantee – that an application that complies with the Oracle Solaris ABI developed and built on an earlier version of Oracle Solaris should run unmodified on Oracle Solaris 11 on the same architecture. In order to provide this guarantee, the application must meet the following restrictions:

- The application must be dynamically linked - neither the application nor any of its component libraries should statically link any of the Oracle Solaris system libraries.
- The application must use only API functions that are documented as committed, standard or stable. Interface stability is described in the attributes(5) man page.
- The application must not use uncommitted, unstable or evolving API interfaces.
- The application must not use **`private`** or undocumented interfaces.

Oracle Solaris provides a number of tools to check an application's compliance against the Oracle Solaris ABI.

On Oracle Solaris 10 the `appcert` and `apptrace` tools can be used to perform static and dynamic checking of an application to test for ABI compliance. This can be done a system running Oracle Solaris 10 before testing the application on Oracle Solaris 11. There is a new tool for Oracle Solaris 11 to check for application compatibility. The Preflight Application Checker tool can be downloaded[4].The package includes documentation which describes how to check an application for compatibility with Oracle Solaris 11.

Oracle Solaris Zone Support

This section will describe the changes brought by Oracle Solaris 11 that software developers will need to be aware of when migrating their applications from Oracle Solaris 10 to Oracle Solaris 11..

Oracle Solaris Zones (also known as Oracle Solaris Containers in Oracle Solaris 10) are lightweight virtual machines that isolate user-level workloads on Oracle Solaris systems. They are based on the Oracle Solaris Zones technology introduced in Oracle Solaris 10. Oracle Solaris Zones differ markedly from virtual machines created with other virtualization technologies such as VirtualBox, and Oracle VM Server for SPARC and x86 in that Oracle Solaris Zones do not rely on hypervisors to provide abstracted hardware resources and isolate themselves from the native host system and each other.

[4] http://www.oracle.com/technetwork/server-storage/solaris11/downloads/preflight-checker-tool-524493.html

Oracle Solaris Zones are built into the Oracle Solaris kernel and many of the kernel's subsystems are zone-aware (i.e., they associate their abstractions with zones and base decisions in part on such associations). Consequently, processes executing within a zone experience little or no overhead (a high estimate is 5% of total execution time) and thus come close to achieving bare-metal performance. Furthermore, the lack of overhead makes Oracle Solaris Zones highly scalable: Even low-end consumer desktops are capable of running dozens of zones at a time. The noticeable lack of execution overhead experienced by zoned processes, the relative ease of zone creation and management, and the maturity of Oracle Solaris Zones technology make Oracle Solaris Zones the most popular virtualization technology supported on Oracle Solaris 10 and Oracle Solaris 11..

However, zones have a few well-known limitations. Furthermore, not all Oracle Solaris kernel subsystems are zone-aware, which limits the kinds of resources that can be fully isolated and supported within zones. Finally, ordinary zones cannot host user environments from non-native, major Oracle Solaris versions. The last limitation is largely eliminated through the use of “branded” zones, which will be discussed in the next section.

Oracle Solaris Zone Features Removed from Oracle Solaris 11

Oracle Solaris 10 Update 4 introduced the concept of branded zones, allowing non-global zones to run a different operating environment than the global zone.

The following brands of non-global zones are no longer offered in Oracle Solaris 11 :

- Oracle Solaris Containers for Linux Applications (“lx”)
- Oracle Solaris 8 Containers brand (“solaris8”)
- Oracle Solaris 9 Containers brand (“solaris9”)

Dependencies of Oracle Solaris 11 Zones

Oracle Solaris 11 Zones are delivered using the new Image Packaging System (IPS). This means that there is no longer any need to choose between the “whole root zone” and “sparse root zone” models available in Oracle Solaris 10. With Oracle Solaris 11 Zones administrators get the best of both worlds and are able to configure zones to match specific application requirements. For example, the personality of a sparse root zone can be duplicated in Oracle Solaris 11 Zones using a combination of ZFS file system, the new IPS, and the use of loopback file system mounts. This configuration provides the ability to precisely control installed packages and the "de-duplication" of data. Both of these capabilities act to significantly reduce the impact of multiple zone installations on a system. As a result, Oracle Solaris 11 Zones are now even more complete and configurable.

- The zone root must be a ZFS dataset, which means it is either a ZFS volume or ZFS file system. In particular UFS is not supported anymore.
- At the time of writing, on-disk IPS repositories are not integrated in Oracle Solaris 11 , which means that network access for the Host system is required to install an Oracle Solaris 11 Zone.

New Features of Oracle Solaris 11 Zones

This section organizes the new features of Oracle Solaris 11 Zones in terms of their impact on ISVs.

Features with High Impact to ISVs

- Non-global zones

  The delivery of non-global zones in Oracle Solaris 11 has been modified. As with the global zone, the system software packages within a non-global zone are managed by the Image Packaging System ( ), the new software lifecycle management framework in Oracle Solaris 11.. As with Oracle Solaris 10, certain packages within the non-global zone much be kept in sync with the global zone. IPS handles this seamlessly to keep the correct packages in sync automatically.

  The following characteristics of non-global zones should be noted:

  - Uses IPS to manage software packages
  - All required software is installed in the private file system of the zone
  - With IPS, updating Oracle Solaris Zones is a trivial operation, and in most cases, it happens automatically during a package update of the global zone. As in the case with the global zone, a new zone BE is created for each non-global zone during the update for updates that require a system reboot. These zone BEs are child BEs of the parent BE in the global zone.
  - Only minimal system software is installed in the zone when it is created. Any additional packages the zone requires must be added after the zone is first booted through the IPS commands.

- Built-in network virtualization and resource management

  Oracle Solaris 11 introduces a new network stack architecture, previously known as "Crossbow". This new architecture provides highly flexible network virtualization through the addition of Virtual NICs, which are tightly integrated with zones. In addition, the new architecture introduces the ability to perform resource management via bandwidth and flow control. The new architecture provides the resource control, performance, and network utilization needed to achieve true OS virtualization, utility computing, and server consolidation.

- As a result, these services can now be run inside a zone:
- DHCP client
- DHCP server
- Routing daemon
- IPsec
- IPfilter
- IP Multipathing (IPMP)
- ndd commands
- ifconfig with set or modify capabilities (usage of dladm and ipadm is recommended)
- Oracle Solaris 10 Zones on Oracle Solaris 11

Please refer to the section entitled, "Oracle Solaris 10 Zones on Oracle Solaris 11.."

Features with Medium Impact to ISVs

- Physical to Virtual (P2V) migration

P2V is the capability allowing the migration of an existing physical Oracle Solaris 10 system image to an Oracle Solaris 11 environment. The system image being installed must not be newer than the host OS release.

- Virtual to Virtual (V2V) migration

Provide the ability to migrate an installed non-global zone from one machine to another using the zoneadm attach/detach sub-commands.

- Update-on-Attach

Feature by which an Oracle Solaris Zone being attached to a new system containing only newer dependent packages/patches (higher revision numbers) than those the zone depended on in the source system, is updated to match the higher revision packages on the new system. In other words, the zone to be attached is upgraded to the package level of the new system. This update is done by providing the -u or -U flags to the zoneadm attach command:

```
# zoneadm -z myzone attach -u
```

- Moving and cloning zones

  - Moving—Enables a zone to be relocated by changing the zonepath property.
  - Cloning—Enables a zone to be quickly copied and provisioned, leveraging ZFS cloning and snapshotting capabilities.

- Exclusive IP stacks

Provide the option for a zone to have its own IP stack so that the zones IP routing, ARP, IPsec, IP Filter, and other configuration are completely disjointed from those configurations for other zones. Enables separation without network security issues for customers that connect separate (V)LANs to

separate zones. Allows a zone to change its own IP address. Exclusive IP stacks used to require a dedicated physical port, however in Oracle Solaris 11 this requirement is negated by the implementation of VNICs.

- Privileges for zones
  Provide the ability to specify the set of privileges that all processes running in a Zone are limited to. This works both ways and can allow DTrace to be run inside a Zone, or prevent a Zone from sending raw ICMP packets.
- Delegated administration for zones
  Integration of zones administration with Role Based Access Control (RBAC). This new feature allows administrators to configure a zone to enable an individual user or role to perform administrative tasks in the zone.

Features with Limited or No Impact on ISVs

- Renaming of zones.
- Default route for zones.
  Allowing the default route for a zone's interface that uses a shared IP stack to be optionally specified using a new default router property, `defrouter`.
- Enhancements to Zones Resource Management
  Although zones are fairly easy to configure and install, many users have difficulty setting up a good Resource Management configuration to accompany their zone configuration. Oracle Solaris 11 introduces several enhancements that provide a simple, tightly integrated experience for configuring zones (zones with RM).
- New arguments for zone boot
  Enabling usage of standard boot arguments when starting zones.
- InfiniBand support
- Enabling processor sets within a zone.
  Creation of a new privilege `PRIV_SYS_RES_BIND` that allows a process to bind processes to processor sets. This privilege can be assigned to a zone, although it will not be assigned by default. This new privilege is a subset of `PRIV_SYS_RES_CONFIG`, so only having `PRIV_SYS_RES_CONFIG` will still allow a process to bind processes to processor sets.
- Parallel Patching of Zones
  Improves the performance of patching tools by allowing parallel patching of multiple zones on the same system. This feature only applies to Oracle Solaris 10 Zones on Oracle Solaris 11..

Oracle Solaris 10 Zones on Oracle Solaris 11 Branded Zones (BrandZ) provide the framework to create zones that contain alternative sets of runtime behaviors. Brand can refer to a wide range of operating environments. For example, in this case the non-global zone can emulate the Oracle Solaris 10 Operating System. Branded zones achieve this by emulating the non-native operating system's system calls (syscalls). Syscalls constitute the sole interface between user environments and kernels;

therefore, if a branded zone emulates syscalls such that they have the same side effects as the system calls of a particular OS (e.g., Oracle Solaris 10), then processes running within the zone will act as though they are running on the targeted OS. A branded zone is the collection of support libraries, support hooks, and auxiliary data files that make emulating the zone's targeted OS possible. Brands are named after the operating systems whose syscalls they emulate. For example, there are Oracle Solaris 8 and Oracle Solaris 9 brands on Oracle Solaris 10 that allow Oracle Solaris 10 systems to host Oracle Solaris 8 and Oracle Solaris 9 user environments, respectively. Zones that host native Oracle Solaris user environments (i.e., zones that lack syscall emulation) are non-global zones.

About the Former Oracle Solaris 10 Brand

The Oracle Solaris 10 non-native zone, described in the Oracle Solaris 10 (5) man page, is a complete runtime environment for Oracle Solaris 10 applications on SPARC and x86 machines running the Oracle Solaris 10 operating system. The brand is supported on all sun4v, sun4u, and x86 architecture machines that Oracle Solaris 11 has defined as supported platforms. The brand supports the execution of 32-bit and 64-bit Oracle Solaris 10 applications. The term "Branded" is not anymore used to refer to Oracle Solaris 10 Zones on Oracle Solaris 11..

The SVR4 (System V Release 4) package metadata is available inside the zone, and the package and patch commands work correctly. Because the zones are whole root zones, all packaging and patch operations will be successful, although the kernel components of the package or patch are not used. For information on SVR4 packaging used in Oracle Solaris 10 native zones, see Chapter 24, "About Packages and Patches on an Oracle Solaris System With Zones Installed (Overview)" and Chapter 25, "Adding and Removing Packages and Patches on an Oracle Solaris System With Zones Installed (Tasks),"[5].. This is the Oracle Solaris 10 version of the guide. The Oracle Solaris 10 zones on Oracle Solaris 11 support the whole root non-global zone model. All of the required Oracle Solaris 10 software and any additional packages are installed into the private file systems of the zone.

Oracle Solaris 10 Zone on Oracle Solaris 11 Creation and Installation

Full documentation on how to create and install an Oracle Solaris 10 non-native zone on Oracle Solaris 11 Zone can be found in the System Administration Guide: Virtualization Using the Oracle Solaris 11 operating system. What follows is a brief overview of the process and example commands.

The brand includes the tools required to install an Oracle Solaris 10 system image into a non-global zone. You cannot install an Oracle Solaris 10 zone on Oracle Solaris 11 directly from Oracle Solaris 10 media. A physical-to-virtual (P2V) capability is used to directly migrate an existing Oracle Solaris system into a non-global zone on a target system. The brand also supports the tools used to migrate an Oracle Solaris 10 native zone to an Oracle Solaris 10 non-global zone that were introduced in Oracle Solaris 10 08/2010. To use the Oracle Solaris 10 zone on Oracle Solaris 11 on your system, you must install the system/zones/brand/s10 package on your target (Oracle Solaris 11) system.

[5]System Administration Guide: Oracle Solaris Containers-Resource Management and Oracle Solaris Zones, part number 821-1460

There are three scenarios for creating an Oracle Solaris 10 zones on Oracle Solaris 11 :

- Creating an Oracle Solaris 10 Zone on Oracle Solaris 11 based an existing Oracle Solaris 10 system or Flash Archive without any non-global zones configured (P2V).
- Creating an Oracle Solaris 10 Zone on Oracle Solaris 11 based on a single Oracle Solaris 10 native zone. (V2V)
- Creating an Oracle Solaris 10 Zone on Oracle Solaris 11 based on a system with Oracle Solaris 10 native zone (P2V and V2V).

Creating and installing an Oracle Solaris 10 Zone on Oracle Solaris 11 based on an Existing Oracle Solaris 10 System or Oracle Solaris 10 Flash Archive Image.

- Access the source system and gather the following information since you will need this when you create the Oracle Solaris 10 Zone one on the Oracle Solaris 11 target system.
  - Host ID (hostid(1))
  - RPC domain name (domainname(1M))
  - Root Password
  - Network utilized on the system
  - File systems mounted
  - Amount of local disk storage used by the existing system.
  - Examine the contents of /etc/system
- Create the Oracle Solaris Flash archive image of the source system.
- Transfer the Oracle Solaris Flash archive image to the target system.
- On the target system, create the zone via zonecfg(1M) command with the Oracle Solaris 10 Zone and use the source systems flash archive image as the install image.

```
target# zonecfg -z my-zone
my-zone: No such zone configured
Use 'create' to begin configuring a new zone.
 zonecfg:my-zone> create -t SUNWsolaris10
 zonecfg:my-zone> set zonepath=/zones/myzone
…
```

- Install the zone using zoneadm install command:

```
zoneadm -z my-zone install -a <path_to_the_Oracle Solaris 10
image flar> -u
```

Creating an Oracle Solaris 10 Zone in Oracle Solaris 11 based on a Single Oracle Solaris 10 Native Zone.

- Use the virtual-to-virtual (V2V) process to migrate an existing zone on your Oracle Solaris 10 system to an Oracle Solaris 10 Zone on Oracle Solaris 11. The V2V process is described at a high level in the steps below. The V2V process for migrating an Oracle Solaris 10 non-global zone to an Oracle Solaris 10 Zone on Oracle Solaris 11 supports the same archive formats as the P2V process. This process uses the zoneadm attach subcommand, which is the existing interface for migrating zones from one system to another.
- Save the zone configuration information on the source system (zoneadm -z <zone-name> info)
- Halt the source zone.
- Create a cpio archive of the zone.
- Transfer the archive to the target Oracle Solaris 11 system
- On the target Oracle Solaris 11 system recreate the zone via zonecfg command with the Oracle Solaris 10 Zone
- Attach the zone from the archive that was created on the source system, with the archive transferred into the /zones directory on the destination system:

```
target# zoneadm -z my-zone attach -a <path_to_the zone_archive>
```

Once the zone installation has completed successfully, the zone is ready to boot

Creating an Oracle Solaris 10 Zone on Oracle Solaris 11 based on an System with Oracle Solaris 10 Native Zone (P2V and V2V).

Using the steps above, you can migrate an Oracle Solaris 10 system with multiple zones configured individually.

Compatibility Issues That Could Prevent an Application from Running in a Oracle Solaris Zone

The following sections will guide you through possible compatibility issues that you may encounter deploying and running your application in a non-global zone. These issues are consistent unless otherwise stated, for all Oracle Solaris Zone technologies, which include Oracle Solaris 10 and Oracle Solaris 11 non-global zones and Oracle Solaris 10 Zone on Oracle Solaris 11 feature of Oracle Solaris 11. Chapter 7 of the Oracle Solaris Zones: Resource Management and Oracle Solaris Zones Developer's Guide documents these restrictions.

Each non-global zone has a security boundary around it. The security boundary is maintained by:

- Reduced process privileges,
- Resource and service virtualization, and
- Inter-zone communication via network only.

Reduced Process Privileges

The Oracle Solaris 10 OS introduced the process Rights Management, which extends the Oracle Solaris process model with privilege sets and uses these privileges sets to control the process access of system resources and kernel services. A process can be either of the following:

- Privilege Aware (PA): completely ignores the effective UID.
- Not Privilege Aware (NPA): behaves almost exactly like a traditional process.

If a process is PA, the kernel only checks the process privileges for system operations. If a process is NPA, the kernel checks both UID and process privileges. A process can attempt to become NPA using setpflags(2).

Each privilege set contains zero or more privileges. Each process has four sets of privileges. One of the privilege sets, the Effective (E) privilege set, determines whether a process can use a particular privilege.

These four privilege sets are:

- E, the Effective set: The set of privileges currently in effect
- P, the Permitted set: The maximum set of privileges for the process
- I, the Inheritable set: The set of privileges inherited on exec(2)
- L, the Limit set: The upper bound of the privileges a process and its offspring can obtain

Oracle Solaris 11 defines a set of 50 privileges. privileges(5) lists these privileges and their definitions. ppriv(1) can be used to inspect or modify process privilege sets.

All processes running in a non-global zone have reduced privileges. That means all processes in a non-global zone are constrained by the privilege sets that are assigned to them when the process is created. Because of the reduced privileges of a process in a non-global zone, certain system calls may return errors. In most cases, EPERM will be returned for a process that does not possess the privilege. Some system calls that check cpc_cpu or net_rawaccess return EACCESS.

It should be noted that only privileged applications in a non-global zone are affected by reduced process privileges. A traditional non-privileged application possesses only five basic privileges. These five basic privileges are available to the non-global zones.

Libraries

The API that the following list of libraries provide, are not supported in a zone. The shared objects are present in the zone's /usr/lib directory, so no link time errors will occur if your code includes references to these libraries.

- libcfgadm(3LIB) – configuration administration library
- libdevinfo(3LIB) – device information library
- libpool(3LIB) - pool configuration manipulation library
- libkvm(3LIB) - Kernel Virtual Memory access library
- libtnfctl(3LIB) - TNF probe control library
- libsysevent(3LIB) - system event interface library

Networking

If a non-global zone is configured without the exclusive-IP stack property, the following networking restrictions apply to the zone:

- IPQoS and IPsec configurations, ipqosconf(1M) and ipsecconf(1M), can only be done in the global zone. A zone-specific configuration can be created by specifying a zone's IP address to the configuration.
- Raw access to layers below the transport layer (for example, IP, ARP, and DLPI to the link layer) is not allowed in a non-global zone. Thus, using DLPI to directly communicate with a link layer (NIC device driver) will result in an error. snoop(1M) will not work in a non-global zone because it uses DLPI to directly access interface drivers.
- The following networking features remain as system-wide features that can only be configured by global administrator:
  - Routing
  - IP Multipathing (IPMP)
  - Mobile IP
  - DHCP Client
  - Network Cache and Accelerator (NCA)
  - Networking tuning via /etc/system and ndd(1M)
  - IP Filter

Devices

- Devices, in general, are shared resources in a system. To make devices available in a zone, therefore, requires some restrictions so the principle of zone isolation will not be compromised. The /dev directory is heavily restricted because of the following constraints:
- Devices that expose system data are only available in the global zone. Examples of such devices are kmem(7D), ksyms(7D), kmdb(7D), trapstat(1M), lockstat(7D), and so on.
- The /dev name space consists of symbolic links (logical paths) to the physical paths in /devices. The /devices name space, which is only available in the global zone, reflects the current state of attached device instances created by the driver. Only the logical path /dev is visible in a non-global zone.
- The global administrator uses zonecfg(1M) to specify the devices to appear in a particular zone. The number of /dev entries in a non-global zone is significantly less than the number of /dev entries in a global zone. The global administrator uses the add device sub-command of zonecfg to include additional devices in a zone.
- The zone administrator can change device permissions, but cannot create new entries.
- Device number is a system-wide property. A system call -- for example, mknod(2) -- that creates a special file mapping to a particular device number will return an error.
- Utilities that perform hardware configuration or change the /dev entries will not work in a zone. These utilities include:
  - add_drv(1M)/rem_drv(1M)
  - modload(1M)/modunload(1M)
  - autopush(1M)
  - cfgadm(1M)
  - devfsadm(1M), drvconfig(1M), disks(1M), tapes(1M), ports(1M), and devlinks(1M)

Differences Between an Oracle Solaris 10 Zone and an Oracle Solaris 10 Zone on Oracle Solaris 11

Networking

The following list identifies Oracle Solaris 10 networking features that are either not supported or are different in an Oracle Solaris 10 Zone on Oracle Solaris 11.

- Mobile IP is not supported. This feature is not available in the Oracle Solaris 11 system.
- Automatic tunnels using the atun STREAMS module are not supported. In an Oracle Solaris 10 Zone on Oracle Solaris 11 , an autopush configuration will be ignored when the tcp, udp, or icmp sockets are open. These sockets are mapped to modules instead of STREAMS devices by default. To use autopush, explicitly map these sockets to STREAMS-based devices by using the soconfig(1M)

and sock2path(4) utilities. In an Oracle Solaris 10 Zone on Oracle Solaris 11 , you must install the patch to support /dev/net links in the Data Link Provider Interface (DLPI) library, which is described in the libdlpi(3LIB) man page. Applications that do not use either the patched libdlpi or libpcap versions 1.0.0 or higher libraries will not be able to access /dev/net links.The following ndd tunable parameters are not supported in an Oracle Solaris 10 Zone on Oracle Solaris 11 :

- ip_squeue_fanout
- ip_soft_rings_cnt
- ip_ire_pathmtu_interval
- tcp_mdt_max_pbufs

- Because IP Network Multipathing (IPMP) in Oracle Solaris 10 Zones is based on the Oracle Solaris 11 operating system, there are differences in the output of the ifconfig(1M) command when compared to the command output in the Oracle Solaris 10 operating system. However, the documented features of the ifconfig command and IPMP have not changed. Therefore, Oracle Solaris 10 applications that use the documented interfaces will continue to work in Oracle Solaris 10 Zones on Oracle Solaris 11 Express without modification. The following example shows ifconfig command output in an Oracle Solaris 10 Zone on Oracle Solaris 11 for an IPMP group ipmp0 with data address 198.162.1.3 and the underlying interfaces e1000g1 and e1000g2, with test addresses 198.162.1.1 and 198.162.1.2, respectively.

```
% ifconfig -a
e1000g1:flags=9040843<UP,BROADCAST,RUNNING,MULTICAST,
DEPRECATED,IPv4,NOFAILOVER> mtu 1500 index 8 inet 198.162.1.1
netmask ffffff00 broadcast 198.162.1.255  ether
0:11:22:45:40:a0
e1000g2:flags=9040843<UP,BROADCAST,RUNNING,MULTICAST,
DEPRECATED,IPv4,NOFAILOVER> mtu 1500 index 9 inet 198.162.1.2
netmask ffffff00 broadcast 198.162.1.255
ether 0:11:22:45:40:a1
ipmp0: flags=8011000803<UP,BROADCAST,MULTICAST,IPv4,
FAILED,IPMP> mtu 68 index 10 inet 198.162.1.3
netmask ffffff00 broadcast 198.162.1.255
groupname ipmp0
```

- Unlike the display produced on an Oracle Solaris 10 system, the ifconfig command in an Oracle Solaris 10 Zone on Oracle Solaris 11 does not show the binding of the underlying interfaces to IP addresses. This information can be obtained by using the arp command with the an options.
- If an interface is plumbed for IPv6 and address autoconfiguration succeeds, then the interface is given its own global address. In Oracle Solaris 10, each physical interface in an IPMP group will have its own global address, and the IPMP group will have as many global addresses as there are interfaces. In an Oracle Solaris 10 Zone on Oracle Solaris 11 , only the IPMP interface will have its own global address. The underlying interfaces will not have their own global addresses.

- Unlike the Oracle Solaris 10 operating system, if there is only one interface in an IPMP group, then its test address and its data address cannot be the same. See the arp(1M) and ifconfig(1M) man pages, and IP Network Multipathing in Exclusive-IP Zones. Zone Root File System

In Oracle Solaris 10, when a zone is created, two options are available to create the root file system of the zone, the Sparse Root and Whole Root models. The Whole Root model (specified via zonecfg(1M) create -b subcommand) provided the maximum configurability by installing all of the required and any selected optional Oracle Solaris software packages into the private file systems of the zone. The advantages of this model include the ability for zone administrators to customize their zone's file-system layout (for example, creating a /usr/local) and add arbitrary unbundled or third-party packages. The disadvantages of this model include the loss of sharing of text segments from executables and shared libraries by the virtual memory system, a much heavier disk footprint (approximately an additional 2 Gigabytes) for each non-global zone configured as such and increase time to install the zone and install patches since the patch files have to be copied to the zone's root FS. The Sparse Root model (default) optimizes the sharing of objects by installing only a subset of the root packages (those with the pkginfo(4) parameter SUNW_PKGTYPE set to root) and using read-only loopback file systems to gain access to other files. By default with this model, the directories /lib, /platform, /sbin and /usr are mounted as loopback, read-only file systems. The advantages of this model are greater performance due to the efficient sharing of executables and shared libraries, a much smaller disk footprint for the zone itself and reduced patch time. The sparse-root model only requires approximately 100 Mbyte of file system space for the zone itself. In addition to Sparse Root and Whole Root model, Oracle Solaris 10 gives administrators the option of creating the zone root on a variety of FS, (UFS, ZFS and NFS). The Oracle Solaris 10 Zone on Oracle Solaris 11 feature supports the whole root zone model. All of the required Oracle Solaris 10 software and any additional packages are installed into the private file systems of the zone. The non-global zone must reside on its own ZFS dataset; only ZFS is supported. The ZFS dataset will be created automatically when a zone is installed or attached. If a ZFS dataset cannot be created, the zone will not install.

Other Restrictions

- The mdb utility is not fully functional when used in the global zone to examine processes executing within an Oracle Solaris 10 Zone for Oracle Solaris 11. Some symbols will not be resolved. This utility can be used from within an Oracle Solaris 10 Zone on Oracle Solaris 11.
- The zfs recv command to receive a snapshot cannot be used inside an Oracle Solaris 10 Zone on Oracle Solaris 11.
- A /dev/sound device cannot be configured into the Oracle Solaris 10 Zone on Oracle Solaris 11.

Command Line Modernization

This section looks at changes to the command line and to the environment experienced by the end user, that have been implemented from Oracle Solaris 10 to Oracle Solaris 11. Many ISV applications are launched from shell scripts and/or require the setting of the PATH and other environment variables. Assumptions about the environment available at the command and in which the application is being launched are based on prior experience with earlier Oracle Solaris releases. There are significant changes to the command line environment in Oracle Solaris 11 and this section sets out to illustrate these changes and looks when/if these changes may come into play when migrating applications from Oracle Solaris 10 to Oracle Solaris 11. The section also details best practices that can be used to minimize the changes required when updating applications and/or documentation for Oracle Solaris 11.

Oracle Solaris 11 has a modernized system environment that contains a mixture of Oracle Solaris and GNU user-level command functionality for users, developers, and system administrators. This combination is necessary to support existing customers and applications and provide a more familiar environment for Oracle Linux users and other users and developers familiar with a GNU environment.

PATH	DESCRIPTION
/etc	Legacy smbolic links have been removed
/usr/bin	default path for command-line utilities
/usr/old	removed
/usr/sunos/bin	path for classic (Sun OS 4) command-line utilities
/usr/gnu/bin	path for GNU command-line utilities
/usr/xpg4/bin	path for versions of command-line utilities (required for UNIX98 compliance)
/usr/xpg7/bin	(proposed) path for versions of command-line (utilities required for UNIX08 compliance)
/sbin	Symbolic link to /usr/bin
/usr/sbin	system administration utilities
/usr/ucb	BSD compatibility utilities (optional!)
/usr/5bin	Deprecated directory. Symbolic link to /usr/bin
/usr/ccs	Deprecated directory. Symbollic link to /usr. C compilation commands and system utilities.
usr/local/bin	Common install location for unbundled GNU utilities

PATH	DESCRIPTION
/usr/old	Utilities that are being phased out (currently empty)
/usr/has/bin	Classic utilities from /usr/bin that have been replaced by GNU equivalents
/usr/sfw/bin /opt/sfw/bin	GNU utilities from the SFW consolidation Oracle Solaris 10, optional location for GNU utilities from the SFW consolidation. Not maintained in Oracle Solaris 11

The Default Path

Given a new user on Oracle Solaris 11 with all variables set to the installation defaults, the following section explains what will change from a similar situation on an Oracle Solaris 10 system.

The default PATH for an Oracle Solaris 11 user may depend on how that user is created initially. There are currently three likely scenarios:

- The user was created using useradd or SMC and has the standard PATH=/usr/bin
- The user has was created at install time and has the extended default PATH=/usr/gnu/bin:/usr/bin
- The user was created at install time and has the root role with PATH=/usr/gnu/bin:/usr/bin:/usr/sbin:/sbin

The PATH variable for existing users can be customized through the.profile and .*rc files found in the user's home directory in the same way as for Oracle Solaris 10. The default PATH for new users can also be modified.

Currently a user created at install time on Oracle Solaris 11 (with or without the root role) has the path `/usr/gnu/bin` prepended to their PATH. This has the effect of providing the user with the gnu versions of popular utilities such as tar, chroot and grep which are usually found in /usr/bin or `/usr/sbin`. The Oracle Solaris versions of these utilities are still available but would need to be run using a full path such as /usr/bin/tar or /usr/sbin/chroot. This may cause issues with users expecting Oracle Solaris behavior from these tools.

`/usr/gnu/bin` also includes a couple of commands for which there is no analog in Oracle Solaris (toe, ncurses5-config).

For some of the more popular tools (or tools where a user often requires access to both the gnu and Oracle Solaris versions), the gnu version often lives in /usr/bin but has a 'g' prefix, for example gtar, gmake and ggrep are the gnu versions of tar, make and grep. These have symlinks in /usr/gnu/bin named with the correct names.

A user wishing to have the Oracle Solaris versions of these utilities by default need only remove /usr/gnu/bin from their PATH. They will also have access to some of the gnu versions via the g-prefixed commands.

Changes to Shells and Shell Environments

The shells available to the user in Oracle Solaris 11 along with their version numbers and paths are listed in the following table.

SHELL	VERSION	LOCATION	NOTES
Bourne shell		/usr/has/bin/sh /usr/xpg4/bin/sh	/usr/bin/sh is now linked to /usr/bin/bash
Korn Shell	93	/usr/bin/ksh /usr/bin/ksh93 /bin/sh /usr/bin/sh	ksh and ksh93 are hard-linked Default for users added with useradd
Bourne Again Shell	4.0.28	/usr/bin/bash /usr/gnu/bin/sh /bin/bash	Default for users added at install
zsh	'4.3.10'	/usr/bin/zsh /usr/sfw/bin/zsh	
tcsh	6.17.00 (Astron)	/usr/bin/tcsh	
csh	?	/usr/bin/csh	

In addition there are the usual 'restricted' versions of ksh and bash available (rksh, rbash) as well as the 'profile' shells: pfksh, pfsh and pfcsh.

The important points to note are that the bourne shell is no longer available on the default PATH, and executing /usr/bin/sh will open a new bash shell. Also the Korn Shell has been updated to version 93 and no older versions are available.

The default shell for users created at install time is /usr/bin/bash and for those created using useradd it's /usr/bin/sh.

Note: /bin/sh is linked to a version of ksh93 and not as you would expect to /bin/bash

Changes in the Directory to /usr/ucb

For a live CD installation, the path /usr/ucb does not exist by default. It can be added, along with many of the standard /usr/ucb tools by installing the 'compatibility/ucb' package from the repository. Note though that support for legacy plotters has been removed and this sees the removal of the following files (PSARC/2009/540):

- /usr/ucb/aedplot
- /usr/ucb/atoplot
- /usr/ucb/bgplot
- /usr/ucb/crtplot
- /usr/ucb/dumbplot
- /usr/ucb/gigiplot
- /usr/ucb/hp7221plot
- /usr/ucb/hpplot
- /usr/ucb/implot
- /usr/ucb/plot
- /usr/ucb/vplot
- /usr/ucb/t300
- /usr/ucb/t300s
- /usr/ucb/t4013
- /usr/ucb/t450
- /usr/ucb/tek

Also note that support for generating SunOS 4 compatible device names through the use of the command /usr/ucb/ucblinks has been discontinued (PSARC/2009/346).

The biggest change to /usr/ucb is the removal of the compiler and linker command wrappers and C headers. As of Oracle Solaris 11 , the following wrapper scripts are no longer available:

- /usr/ucb/ld
- /usr/ucb/lint
- /usr/ucb/cc

and the /usr/ucbinclude directory and it's contents have been completely removed. /usr/usrlib is retained (as part of the compatibility/ucb package) for backwards compatibility.

Other than these changes all of the commands available in /usr/ucb in Oracle Solaris 10 continue to be available in Oracle Solaris 11 through the compatibility/ucb package.

Oracle Solaris 11 Packaging Mechanisms

This section provides a brief overview of IPS (Image Packaging System) in Oracle Solaris 11 , particularly within the context of what might be of specific interest to ISVs. It also includes links to other useful material that will help ISVs get up to speed with IPS along with pointers for those that just want to get started up and running with IPS. Package creation will be covered in greater detail under separate cover.

Emphasis will also be placed on:

- The remaining compatibility with existing SVR4 packages
- What's missing from SVR4 in Oracle Solaris 11
- Providing solutions and best practices to help ISVs and IHVs migrate their applications away from SVR4 packaging and onto IPS

Lastly, this section covers the "how-tos" around making IPS packages available to customers. This includes setting up repositories and the on-disk format, which can be used to create executable installers.

Introducing IPS

The Image Packaging System (IPS) is a highly portable and network-centric packaging and delivery system, designed to allow efficient, observable, and controllable transitions between known configurations of software content. First developed as part of OpenSolaris, IPS replaces SVR4 packaging in Oracle Solaris 11 , although some aspects of SVR4 are preserved to maintain some level of compatibility for existing packages.

IPS is a paradigm shift from the legacy packaging system used in versions of Oracle Solaris prior to Oracle Solaris 11 , and may seem a little overwhelming at first. Oracle strongly recommends that all ISVs migrate to this new system as soon as practical, as it will simplify system administrators' packaging and patching duties. That said, it is recognized that you may just want to get up and running with what you have today and deal with this change at a rate at which you are comfortable. To that extent, Oracle Solaris 11 continues to support SVR4 packaging and continues to provide the tools that you are you have previously used to install, manage and maintain SVR4 packages. There are some restrictions and these are discussed in the section "Legacy Packaging".

For more complete information about IPS please visit the online documentation:[6]

[6] http://download.oracle.com/docs/cd/E19963-01/820-6572/820-6572.pdf

One of the key motivations behind the development of IPS was the disconnect between packaging and patching in SVR4. The idea that an arbitrary set of patches could be applied to an arbitrary collection of packages led to the creation of entirely new and unpredicted software configurations and to all of the support implications that that implies. This has resulted in Oracle Solaris 11 being designed with some basic packaging and patching tenets:

- Packaging moves from the notion of individual, file-based packages to the concept of network- or disk-based package "repositories"
- Package manipulation will be more intuitive and granular, and assumes (but does not always require) that most systems have network access
- Patches cease to exist; changes to software are to be folded into newer versions of that software's package, available for install or upgrade as the System Administrator sees fit

To reiterate:

There is no concept of Patching in Oracle Solaris 11 Express and beyond.

For more information on the thinking behind IPS and patching, see the Bart Smaalders blog entry[7].

IPS Packages

In IPS, all packaging data is found in packages and a package is a minimization boundary. There are no patches, only package versions and dependencies. Install-time scripting is no longer supported; instead IPS uses the notion of software self-assembly, whereby environmental considerations determine some aspects of the final configuration of a package.

For those migrating SVR4 packages to IPS, any scripting must be removed. There are alternatives that would allow for similar install-time functionality, if required. The basic approach is described in Constantin Gonzalez's blog post[8].

To reiterate:

There is no pre-install or post-install scripting capability in IPS; other arrangements must be made

Each package (version) is named with a unique FMRI (Fault Management Resource Indicator). A package is defined by a manifest that is made up of a list of actions. IPS defines a limited set of actions[9], including actions for files, directories, dependencies and links.

Dependencies between packages are declared in manifests and are managed by IPS. A dependency can be one of:

[7] http://blogs.sun.com/barts/entry/rethinking_patching

[8] http://constantin.glez.de/blog/2010/08/how-add-pre-post-scripts-ips-packages

[9] http://dlc.sun.com/osol/docs/content/2009.06/IMGPACKAGESYS/actions.html

DEPENDENCY NAME	DESCRIPTION
require	Package must be present at specified version or newer
optional	If installed, package must be present at specified version or newer
exclude	If installed, package must be older than the specified version
incorporate	If installed, package is constrained to the specified version within the specified precision

Packages are installed to locations in the file system called images. In a Full Image, all package dependencies are satisfied from within the image, while a linked image will contain packages whose dependencies are satisfied by packages installed to a full image. An example of a linked image is the set of packages installed, by default, into an Oracle Zone. A live image is a full image that contains a booted operating system.

Installation of Packages

With IPS, packages are installed into an image from package repositories, where a package repository is a location specified by a URI and associated with a package publisher. A publisher can aggregate multiple package repositories (although commonly there is a one-to-one mapping from publisher to repository). An image may be configured to use more than one publisher as a source of packages.

A repository URI may specify either `file` or `http` as the protocol. This allows for the installation of packages from either network-based or file system-based repositories.

As mentioned above, packages are identified fully by an FMRI. The full FMRI consists of the publisher name, the package name and a 4-part version string. When specifying a package, the publisher name and part or all of the version string may be omitted. Only one version of a package may be installed in an image at a given time.

At the core of IPS is what's known as `pkg(5)`. `pkg` is a command and the (5) indicates the man page section (in this case Section 5: Standards, Environments and Macros) that the specification for the pkg command appears in (use `` `man -s 5 pkg` `` to display this man page). `pkg(5)` and a small number of other commands are used to manage all aspects of IPS, from package creation to building a repository.

In addition to pkg commands, the Package Manager GUI tool is provided for most of the pkg-based functionality, should the System Administrator prefer.

Legacy Packaging

In Oracle Solaris 11 , many IPS packages have one or more 'legacy' actions that include a legacy (SVR4) package name (for each supported architecture) associated with the package. For example the 'network/ssh' package has the legacy package names SUNWsshr, SUNWsshu, SUNWsshdu and SUNWsshdr associated with it and these are the names for the equivalent packages on Oracle Solaris 10.

Oracle Solaris 11 includes an IPS package named 'package/SVR4' which includes versions of the SVR4 tools that are found in Oracle Solaris 10 (`pkgadd`, `pkgrm`, `pkgchk`, etc). For SVR4 packages, these tools continue to function as they did in Oracle Solaris 10. For IPS packages with legacy actions, these tools can see those packages but cannot manage them.

The 'package/SVR4' package tools can be used to install, manage and maintain legacy packages in the same way as on Oracle Solaris 10.

As previously mentioned, Oracle Solaris 11 has no concept of patching, and none of the patch management commands familiar to anyone deploying applications on Oracle Solaris 10 are available. Because there is no patching and no patches, there is no `showrev` command. `showrev` is used on Oracle Solaris 10 to provide a list of installed patches, and is often used with regular expressions in scripts to determine the patch level of system or of specific packages. On Oracle Solaris 11 , use the command

```
$ pkg list [pkgname]
```

to determine the version number of an installed package or packages.

If you wish to continue to deliver SVR4 packages, any updates that you make will need to be delivered through updates to those packages.

Converting Packages from SVR4 for use with IPS

`pkg(5)` includes the `pkgsend` command, which is used to publish new packages or new versions of packages to a repository. `pkgsend` can be used in a variety of ways, and in this case we consider its use in generating a manifest from an existing SVR4 packages and then publishing that manifest (along with required files) to an IPS repository.

Generating a manifest from an existing SVR4 package is straightforward, but it only works on SVR4 package directories not on package datastreams (as created by `pkgtrans`). To generate a manifest run:

```
$ pkgsend generate MYpkg > Mypkg.mfst
```

Where `MYpkg` is the path to the directory representing the SVR4 package you wish to publish and `Mypkg.mfst` is the name of the manifest file that you wish to create.

Distributing IPS Packages

As discussed above, IPS packages are installed by users from package repositories that are maintained by package publishers. Repositories can be either network-based or file system-based. Network-based repositories are usually either accessed via the Internet, via a corporate Intranet or via a specific server accessible only by a limited number of other systems (e.g. in a lab or datacenter environment.)

Examples of Internet-accessible network-based repositories include:

- http://pkg.oracle.com
- http://pkg.sunfreeware.com:9000

If Oracle Solaris 11 is used extensively throughout an enterprise, it may be viable to provide a network-based repository on the corporate Intranet. Within a data center or lab, external network access may be restricted, in such a case a network-based repository could be made available within the confines of the lab or datacenter.

The requirements for access to network-based repositories depends heavily on what packages those repositories are publishing. The scenarios described above are common solutions for accessing the contents of the Oracle Solaris 11 release repository, which contains all of the packages that make up the Oracle Solaris 11 distribution.

The alternative to a network-based repository is a file system-based repository. A file system-based repository is used for local installation of IPS packages and a user installing packages from such a repository uses the same set of `pkg(5)` commands, but with alternative command line options (or through some changes to the configuration of `packagemanager`).

POSIX compliant Header Files

29 declarations are now in conformance with the POSIX:2008 or IEEE Std 1003.1-2008 specifications. The interfaces are the same but the function prototypes have changed. If code no longer compiles which uses the 29 functions declared in dirent.h, iconv.h, ndbm.h, stdlib.h, unistd.h, or xti.h , it is recommended to change the code to be in conformance to the latest POSIX specifications and the changed headers. If that is not feasible due to large amounts of code, it is possible to compile using the old style declarations by specifying -D__USE_LEGACY_PROTOTYPES__ on the compile line.

Selected Removed Packages and Frameworks

This section concerns the software that has been removed from Oracle Solaris 11. This includes applications, libraries, middleware, utilities and drivers. Oracle is maintaining a new "Oracle Solaris 11-End of Feature Notices" web page[10] in the Oracle Technical Network (OTN). Please revisit this web

[10]URL: http://www.oracle.com/technetwork/systems/end-of-notices/eonsolaris11-392732.html

page from time to time. It is a living document. This section discusses a selected list of the currently known removals. Alternatives are suggested when appropriate.

Applications and Middleware

- Evolution-jescs has been replaced with Lightning 0.3
- StarOffice is removed. OpenOffice may be installed individually to achieve backwards compatibility.
- Apache 1.3 is replaced by Apache httpd 2.2
- MySQL 4.0, 5.0 is upgraded to MySQL 5.1
- PostgreSQL is removed. Alternatives are Oracle's RDBMS, MySQL or the community version of PostgreSQL.
- Berkeley DB 4.2 has been upgraded to 4.7
- MMS (media-management system) has been discontinued.
- NIS+ has been replaced with LDAP name service

Libraries

- audit_user(4) and getausernam(3bsm) both are replaced by "audit_flags" keyword in user_attr(4) and prof_attr(4)
- getacinfo(3bsm) and audit_control(4) are superseded by use of SMF
- FMLI (Forms and Menu Language Interpreter) has been discontinued
- Graph/Spline functionality is covered by Gnu's plotutils
- iSCSI target Daemon is replaced by COMSTAR
- SEA is replaced by Net-SNMP 5.4.1
- SER is replaced by libsip
- Xsun is replaced by Xorg or Xnewt for SunRay
- Xsun on Oracle Solaris x86 has been replaced by the Xorg X server
- Obsolete @euro locales have been discarded
- Short form locales have also been removed.
- Obsolete ioctl interfaces in mpt driver have been replaced by PSARC/2006/544
- OCF/SCF Smartcard Framework has been removed
- pcmciad and pem are replaced by devfsadm

- X Image Extension (XIE) Library has been removed. Image processing is handled popularly in a variety of libraries and utilities

Utilities

- Linux, Oracle Solaris 8 and Oracle Solaris 9 branded zones are no longer supported.
- /usr/ucb development tools are replaced with the Sun Studio tools.
- audit_control/startup, bsmrecord/conv/unconv have been replaced with auditrecord and SMF
- bdftosnf and showsnf commands are replaced with bdftopcf and showfont
- CD Font Administrator is replaced with GNOME font control panel
- mixerctl is replaced by audioctl
- plotting support is now via gnuplot
- sag(1) is now done via kSar and sar2rddc
- xmh has been superseded by Thunderbird and Evolution
- xorgcfg and xorgconfig have been removed. Xorg auto configuration is now the norm.
- Xprint has been replaced by CUPS and the GNOME printing system

Drivers

The rational for removal of these drivers is simply the fact that the hardware is ancient and no longer relevant.

- Creator & Creator 3D (ffb) SPARC graphics
- Elite3D (afb)SPARC graphics
- Expert3D & XVR-500 (ifb) SPARC Graphics
- m64 SPARC graphics
- SPWR NIC driver
- XVR-600 & XVR-1200(jfb)SPARC Graphics
- CG6 Driver for GX series SBUS Graphics cards
- PCMCIA memory card support
- SBPRO (Soundblaster 16 driver)
- SDCard support for SPARC
- Support for pre-2006 SPARC Workstations

- XVR-4000, XVR-1000, PGX32 graphics cards

Internationalization

Removed @euro locales for territories: ES, DK, AT, DE, GR, IE, FI, FR, IT, NL, PT

Applications with try to switch to an @euro locale for the country codes ES, DK, AT, DE, GR, IE, FI, FR, IT, NL, PT will fail under Oracle Solaris 11.

The Euro became a legal currency January 1, 1999 in the Spain, Denmark, Austria, Germany, Ireland, Finland, France, Italy, Netherlands and Portugal. The Euro replaced national bank notes by July 2002. It has been necessary through this period to have a national formatting locale for the Euro as well a national formatting for the old currency. This problem has been solved by providing the regular locales (example: fr_FR.ISO8859-15) and in addition the "Euro" locales with the "@euro" modifier at the end of the locale name (example: fr_FR.ISO8859-15@euro).

The @euro locales became obsolete by July 2002 since the period of a dual currency ended. The @euro locales for the countries mentioned are not be part of Oracle Solaris 11. The locale that needs to be used instead is the locale without the "@euro" modifier at the end. The behavior of the non @euro locale is identical to the @euro locale since Oracle Solaris 10. Oracle Solaris 9 however applied the national format of the old currency with the non @euro locale. The table below illustrates the different behavior across Oracle Solaris 9 and 10 and Oracle Solaris 11.

ORACLE SOLARIS VERSION	@EURO LOCALE	REGULAR LOCALE
Oracle Solaris 9	national Euro format	national format old currency
Oracle Solaris 10	national Euro format	national Euro format
Oracle Solaris 11	not available! Use non@euro equivalent	national Euro format

A deeper discussion of the @euro locales can be found in the Oracle white paper, “Euro Currency Support in the Oracle Solaris Operating Environment.”

Removal of Short Form Locales

Oracle Solaris 10 has 113 so-called short form locale names being provided and supported at Oracle Solaris such as "ar", "de", "fr_CH". These short form locales have inherent ambiguities in identifying intended territory, code set, or both.

All modern POSIX-like operating systems including Oracle Solaris, AIX, HP-UX, and various Linux distributions provide and support locale names with the following convention:

language_territory.codeset[@modifier]
Where the "language" is a 2-letter language code from ISO 639, the "territory" is a 2-letter uppercase territory code from ISO 3166, the "code set" is one of the widely accepted code set names in the industry, and the optional "@modifier" is an arbitrary string denoting an extra attribute that is specific to the locale such as a collation order.

Oracle Solaris 11 does not offer these short form locales anymore. Applications can still use them with a newly introduced Solaris technology called "Locale Aliasing"[11]. The concise mapping to the ISO compliant locales is being discussed in the `locale_alias(5)`[12] man page of Solaris 11.

Troubleshooting Crashes Caused by Known Coding Issues

Even with the ongoing binary compatibility between Oracle Solaris 10 and Oracle Solaris 11 , developers may find that certain application binaries fail mysteriously on Oracle Solaris 11 , without having exhibited such behavior before. This sections details several cases where original incorrect coding leads to seemingly new failures in Oracle Solaris 11 , due to changes in private, unstable APIs.

Case 1: Segmentation Violations Based on Misaligned Mutexes

- Environment(s):
  - Oracle Solaris 10 systems with patch revisions 137111-01 to 137111-08 (SPARC only)
  - Oracle Solaris 11 (SPARC only)
- Symptoms:
  - Application binaries suddenly fail with only the message "Memory fault".
- Root Cause:
  - Objects of type `mutex_t` and `pthread_mutex_t` must start at 8-byte aligned addresses. Those applications that were not developed in this way can fail in Oracle Solaris 11 for SPARC versions.
- Proper Resolution:
  - Change the source code to align mutexes to 8-byte boundaries and rebuild the binary.
- Workaround:
  - The segmentation violation can be avoided (albeit at some cost of performance) by setting the environment variable `_THREAD_LOCKS_MISALIGNED` to `1`
- More information: Oracle Solaris 10 10/08 Release Notes[13]

[11]URL: http://docs.oracle.com/cd/E23824_01/html/E26033/glmen.html

[12]URL: http://docs.oracle.com/cd/E23824_01/html/821-1474/locale-alias-5.html#REFMAN5locale-alias-5

[13]URL: http://download.oracle.com/docs/cd/E19253-01/820-5245/6nglfuqi8/index.html

Case 2: Segmentation Violations Due to Statically-Linked `libc` Libraries

- Environment(s):
  - Oracle Solaris 10 (possible)
  - Oracle Solaris 11 (likely)
- Symptoms:
  - Application binaries fail with a Segmentation Violation.
- Root Cause:
  - Static linking of applications to libc.a and libthread.a has been disallowed since Oracle Solaris 10. Application binaries created in Oracle Solaris 9 or earlier that do this may fail, and are not guaranteed to work in Oracle Solaris 10 and later. Use of the Oracle Solaris ldd command can be used to identify this situation.
- Proper Resolution:
  - Change the source code build environment to not statically link with these libraries. More information: Rod Evan's blog post[14] provides a good explanation of the technical background

Tools to Manage the Transition to Oracle Solaris 11

Oracle Solaris Studio

Oracle Solaris Studio is a comprehensive tool suite for classic C, C++ and Fortran developers that enables ISVs to create applications for Oracle Solaris 10 and 11. We recommend using the latest version of the product (currently Studio 12.3) as this will contain the latest optimizations and support for the most recent processors, it will also contain feature improvements over the previous versions.

Oracle Solaris Studio OTN page[15] is the landing page for the product download, whitepapers, technical articles, screen casts, webcasts, forums and much more.

Introduction

Oracle Solaris Studio delivers tools that are optimized for the underlying operating system and hardware to help you create high-performance, scalable and reliable applications. Oracle Solaris Studio contains two major suites of tools, a Compiler Suite and an Analysis Suite. The tools within the tool suite are designed to work together to provide an optimized development environment for the development of single, multithreaded, and distributed applications.

[14]URL: http://blogs.oracle.com/rie/entry/static_linking_where_did_it

[15]URL: http://www.oracle.com/technetwork/server-storage/solarisstudio/overview/index.html

Compiler Suite

The Compiler Suite includes the following components: C & C++ Compilers, Fortran Compiler, Debugger, Performance Library.

The C, C++ compilers use advanced code generation technology to help you create the highest performance applications for the latest SPARC and x86-based Oracle Sun servers. The compilers provide a solid foundation for building robust, high-performance parallel code. In addition to supporting the latest language standards, Oracle Solaris Studio software also includes GNU C/C++ compatibility features and is compatible with prior releases from the source and object level. This allows you to seamlessly link objects compiled with older versions of the compiler with objects compiled with newer versions of the compiler.

To take advantage of hardware concurrency in multicore systems, the compilers simplify the creation of parallel applications with autoparallelization features. These features enable the compiler to identify safe and profitable parallelization opportunities in single-threaded code and automatically convert those segments into multithreaded code. In addition, the compilers support the OpenMP 3.0 specification that introduces task-based parallelism.

The compilers in Oracle Solaris Studio include an array of optimization options for increasing application performance. For generating everything from microarchitecture-specific instructions and profile feedback to whole-program optimizations, the compilers provide both a wide selection of individual options and easy-to-use metaoptions for aggressively optimizing application performance.

In addition to the C and C++ Compilers, Oracle Solaris Studio also includes a Fortran compiler. It provides compatibility options for the Fortran77, Fortran90, and Fortran95 standards to support the existing base of codes in the technical market.

The dbx Debugger helps ensure the correctness of your applications. It is an interactive, source-level, postmortem and real-time debugging tool that is available through the command line, the IDE and the standalone graphical debugger (dbxtool).

For ISVs in the technical computing market, Oracle Solaris Studio also delivers a Performance Library that includes advanced numerical solver libraries to help maximize the performance of compute-intensive applications.

Selecting Appropriate Compiler Flags

The compiler can perform a wide variety of optimisations depending on the flags provided to it. The documentation exhaustively covers these flags. In t his document we cover only the most critical points of compiler flag usage.

The first point to observe is that the generation of optimised code requires optimisation flags. So it is recommended that an optimization level of at least -O be used for building any application.

The only situation where an unoptimised build might be used is if the application will be used purely for debug. An unoptimised build will produce a build with full debug capabilities, using optimization may reduce the amount of debug information that is available. The flag to generate debug information

is -g and it is recommended that this flag always be included even for optimised builds, where the flag has minimal impact on performance.

The compiler also supports the flag -fast which is an aggressive optimization flag. It enables a number of optimizations that should result in better application performance. However, it is suggested that the performance under this flag be compared with the performance under -O. If -fast results in faster performance, then determine the optimizations enabled by -fast which contribute to the performance gain, and explicitly use these. The recommendation here is to use only those compiler flags which you both understand and also produce a performance improvement.

64-bit or 32-bit Code

One of the important decisions to make is whether to produce a 32-bit or 64-bit application. Care has to be taken as the change will affect the internal structures in the applications, and consequently not all applications are ready for 64-bit compilation. Lint can be used for C applications to check for 64-bit portability issues.

The primary motivation for moving to 64-bit applications is to increase the memory that the application can address. The limit for 32-bit applications is 4GB which is insufficient for some of the problem sizes that users encounter today.

On x86 processors there is typically a performance advantage to moving to 64-bit applications. This advantage comes from improvements in the instruction set and improvements in the way functions are called. However, in general 64-bit applications will consume more memory than a 32-bit equivalent, so not all applications will see the performance gain. If the data structures within the application are dominated by pointer variables, or variables of type long, the overhead of the memory increase may outweigh the performance gain from the instruction set improvements.

32-bit SPARC applications already have most of the optimisations that x86 obtained at its 64-bit transition. Hence 64-bit applications on SPARC will usually see a slight decrease in performance because of the additional memory footprint.

Advanced compiler Optimizations

There are two compiler optimizations that increase compile time, but will often result in useful performance gains. These are cross-file optimization and profile feedback, in many ways the two options are complementary.

Cross-file optimization requires the use of the flag -xipo both when compiling the file and when linking the application. It causes the compiler to take a second optimization pass at link time looking for additional places where performance could be gained. This has the downside of increasing link time, but it has the advantage that it can look across all the object files searching for performance opportunities, it is particularly useful when the source code is distributed across many files.

Profile feedback is an optimization that particularly targets applications which are dominated by branches. It is important because branch-dominated codes are typically hard to optimize in other ways. It is controlled with the flag combination -xprofile=collect:<file> and -xprofile=use:<file>. The first

of these flags will cause an instrumented executable to be generated. The instrumented executable can be run multiple times with different representative workloads and it will accumulate data about the runtime behavior of the application. This data can then be used in the second compiler pass with the flag -xprofile=use:<file>. The second pass allows the compiler to make optimization decisions using the data accumulated during the instrumented run. This allows the compiler to make better decisions about code layout or function inlining which typically results in performance gains for all workloads.

Using the C++ Standard Library

The C++ compiler supports, out-of-the-box, three versions of the C++ Standard Library. Mixing different versions of the STL within the same application will cause undefined behavior Hence it is important to ensure that the application and all its libraries standardize on a single version.

The three versions are the default STL, stlport4, and the Apache STL. The default STL predates the C++ standard and is missing a number of features, however it exists on all systems. It can also be less performant than the alternatives.

The compiler includes stlport4 as an alternative. This library can be selected using the flag -library=stlport4. It is typically faster than the default STL and supports a more complete set of features. The library is shipped with the compiler and it will need to be distributed with the application.

The Apache STL is available under the compiler flag -library=stdcxx. It also tends to be faster than the default STL and supports a more complete set of features. The Apache library is distributed as part of Solaris.

Linking Applications

Applications should be linked by invoking the compiler, the linker should never be called upon directly to perform the linking. This is because the compilers know which support libraries are required for an application to work correctly, and they also know which order the libraries should be linked in.

Linking pure language applications should be performed using the appropriate compiler, for example cc should be used to link pure C applications. Mixed language applications need to either use the Fortran compiler if the code is a mix of C and Fortran, or the C++ compiler if the application contains a mix of C++ code with Fortran or C.

Applications that depend on runtime libraries should use the flag -L to specify the location of those libraries at link time, and the flag -R to specify where the libraries will be found at runtime. The token $ORIGIN can be used with the -R flag to indicate the path to the runtime libraries relative to the application. Applications should never rely on the setting of LD_LIBRARY_PATH to function correctly.

Writing portable Applications

To generate portable applications you should build on the oldest build of Solaris that you plan to support with the most recent tools that are supported on that platform. In many cases this will mean

building on Solaris 10 with Oracle Solaris Studio 12.3. The Solaris binary compatibility guarantee ensures that applications that run on older Solaris versions will also run on more recent versions. So an application built on Solaris 10 will continue to run on Solaris 11.

Oracle Solaris Studio enables you to choose the range of processors supported by your application. The default is to generate a "generic" binary which will run on all the processors supported by the compiler release.

However, there are some instances where the application will only be run on a particular processor, or on a processor which supports a particular set of features. An application may obtain a performance advantage by utilising features of the set of processors. The only disadvantage to this leveraging these features is that the application will not run on processors that do not have them.

For x86 processors there is sometimes a performance advantage to using SIMD instructions. 64-bit applications on x86 can assume the presence of SSE2 instruction set extensions, 32-bit applications do not make this assumption. For 32-bit applications there may be a benefit from allowing the compiler to generate SSE2 instructions using the command line flag -xarch=sse2. To get the full benefit of these instructions for both 32-bit and 64-bit applications it is necessary to add the flag -xvector=simd which will cause the compiler to aggressively identify loops in the code where these instructions can be generated.

Recent SPARC processors support the fused multiple accumuolate instruction. This performance a floating point add and multiply in a single step – potentially doubling the floating point capability of the system. To generate these instructions use the compiler flags -xarch=sparcfmaf -fma=fused. As is the case with all floating point optimisations, these flags may have cause a difference in the results. Performing a multiply and add separately will involve two rounding operations when the results are written back to registers. A combined multiply and add operation will only round the final result, not the interim value. This has the potential to cause a difference to the least significant digits of the result.S

Writing Parallel Applications

The Studio compilers support multiple ways of developing multithreaded applications. The tools also support the analysis, optimisation, and debug of multithreaded codes.

The easiest way of producing a multithreaded application from an existing serial code is through automatic parallelisation. This asks the compiler to detect loops in the code where the work can be distributed across multiple threads. This analysis is controlled by the flags -xautopar and -xreduction, information about which loops the compiler has parallelised can be obtained using the flag -xloopinfo.

OpenMP is a more flexible approach to parallelisim. It involves manually inserting directives into the code that tell the compiler how to parallelise the application. OpenMP is a very easy way to generate a parallel application because the compiler does the tricky work of managing the threads and generating the code. The compiler flag -xopenmp controls whether the compiler recognises the openMP directives. The flag -xvpara issues messages indicating any parallelisation issues that the compiler detects.

The third approach to writing parallel code is to use POSIX threads. This mode is fully supported by the tools, and requires no special compiler flags except -mt. The flag -mt indicates that multithreaded code is being generated, and changes the API defined in the header files to be thread safe.

Analysis Suite

The Analysis Suite includes an advanced suite of tools to help ISVs gain increased observability into their applications. The suite includes the following tools: Performance Analyzer, Code Analyzer, Thread Analyzer.

The Performance Analyzer

The Performance Analyzer identifies application performance bottlenecks, by specifying not only which functions, code segments, and source lines have an impact on performance but by also providing the tools necessary to do tuning for optimal performance. From annotated compiler commentary listings in which the compiler indicates a range of information to optimization status and runtime thread performance, users can visualize performance hotspots with the GUI. The performance analyzer tool can be used to profile single-threaded as well as multithreaded applications.

The Performance Analyzer does not require a special build of the application. It will, however, be able to produce more detailed results if the application is built with debug information. Profiling applications is a two step process. The profile is gathered by running the application under the tool collect, for example, collect./a.out. This will generate an experiment repository named test.1.er by default. This experiment can be examined using the analyzer GUI or through the command line tool er_print. The Performance Analyzer can gather data from a running process using collect's -P <pid> option, and is able to generate exact instruction count data using collect's -c on option.

The Code Analyzer

The Code Analyzer helps ensure application stability by utilizing dynamic, static and code coverage analysis to detect application vulnerabilities, including memory leaks and memory access violations. It gives you comprehensive memory error detection to help you improve the reliability of your applications. The Code Analyzer looks at static analysis when you are compiling your application and it looks at dynamic analysis when you are running your application and gives you feedback about where you may have errors. In addition, it looks at code coverage data to give you information about functions that are not covered by your test suite and provides guidance on they type of benefit you could get by covering those functions.

Static error detection is enabled using the compiler flag -xanalyze=code at compile time. This generates a report of the issues detected in the application. This report can be viewed using the Code Analyzer GUI. Static error checking represents only those errors that can be detected by the compiler by a deep analysis of the source code. Many errors cannot be detected by a purely static analysis of the code, hence the Code Analyzer also includes support for runtime analysis.

The Code Analyzer supports code coverage analysis. Code coverage is useful in determining whether a test workload covers all the functions in an application. The tool is named "uncover" as it is more important to focus on reducing the parts of the code that are not covered by tests than it is to examine the parts of the code that are tested. However, code coverage indicates weakness in the test suite, it does not discover potential runtime issues.

The tool discover evaluates the runtime behaviour of an application and detects the common memory access errors. Memory access errors, such as writing past the end of an array, or reading an uninitialised value, represent the most common type of programming, and security error. The detailed analysis that discover performs is impossible for a static analysis tool. One key advantage of discover is that it does not require a special build of the application. Once your application has been compiled you only need to run the command discover on your executable and discover will generate a version of your application which will report any runtime memory access errors that it encounters. The critical point of not requiring a special build of your application is that if you have to perform a special build then you are no longer getting details of the errors that your application encounters, but are getting the errors from an application version built with different optimisations.

The Thread Analyzer

The Thread Analyzer identifies hard-to-detect threading errors before they occur. It can detect potential race and deadlock conditions at runtime, map them to source lines in the application, and then enable the user to view the results by using command-line or graphical user interface (GUI) options. Debugging multithreading programming errors is challenging and the Thread Analyzer helps simplify the process by identifying both deterministic and non-deterministic behavior.

The mechanism for gathering details of threading errors is to run an instrumented version of your executable under collect -r all, and then examine the results under the Thread Analyzer. An instrumented version of the executable can either be built using the compiler flag -xinstrument=datarace, or an existing binary can be instrumented using the command discover -i datarace./a.out.

Oracle Solaris Studio IDE

In addition to the Compiler Suite and the Analysis Suite, Oracle Solaris Studio also offers an integrated development environment specifically geared for C, C++ developers that helps increase developer productivity.

The Oracle Solaris Studio IDE is built on the NetBeans platform and has a variety of advanced features that help increase developer productivity including an intelligent language aware code editor, code completion, code folding, syntax highlighting and much more.

The Oracle Solaris Studio IDE also supports remote development enabling you to create Oracle Solaris applications from non-Oracle Solaris clients. It also has a dynamic class browser, a wizard for creating makefiles and is highly configurable.

Oracle Solaris 11 Preflight Application Checker[16]

The purpose of the "Oracle Solaris 11 Preflight Application Checker" is to ease the application migration from Oracle Solaris 10 to Oracle Solaris 11 Express and Oracle Solaris 11. The "Oracle Solaris 11 Preflight Application Checker" (S11CompatCheckTool) analyzes applications already working on Oracle Solaris 10, and reports known compatibility issues with Oracle Solaris 11.

The purpose of the S11CompatCheckTool is to enable the developers to support their applications on Oracle Solaris 11 both more quickly and with lesser efforts.

Introduction

While it is expected that a large set of applications currently running on Oracle Solaris 10 will continue to run unmodified on Oracle Solaris 11 , there may be few applications which may have some incompatibilities causing them to fail on Oracle Solaris 11. The Oracle Solaris 11 Preflight Application Checker can help the developers, by giving them an estimate of the existing incompatibilities and the changes required to support the application on the new platform. By using Oracle Solaris 11 compatibility check tool on the Oracle Solaris 10 based development or test deployment systems, developers can get insight into the application's compatibility issues on Oracle Solaris 11. This is achieved by executing a combination of "Source Code analysis", "Static Binary analysis" and "Runtime analysis" modules of S11CompatCheckTool, on the applications already available on Oracle Solaris 10.

Binary / ELF Analyzer:

All interfaces in Oracle Solaris are versioned to ensure upward compatibility, and labeled as public or private. Public interfaces are those that are intended for use by applications or middleware running on top of Oracle Solaris, while private interfaces are those only intended to be used by other Oracle Solaris interfaces.

Although the most problematic cases of binary incompatibility result from the use of private symbols, other issues can arise from unstable practices. This includes static linking of Oracle Solaris shared object to the application at compilation time, since Oracle Solaris interfaces may change over time. As these interfaces evolve, the interactions between the shared object being statically linked and other Oracle Solaris libraries become at risk for binary incompatibility as well. Therefore, in the compatibility check phase, when static linking is noted in the profiles of the objects, the compatibility checking tool will report this as dangerous. In addition, heuristics are used for checking partial static linking by looking at particular symbols.

During the initial phase of binary scan, all bindings and associated symbols that are used by the application are extracted. S11CompatCheckTool uses the user supplied application environment variable LD_LIBRARY_PATH to determine the libraries being used by the application during runtime, which in turn is parsed to find out, path names of the objects being examined and symbol bindings into Oracle Solaris shared libraries. Another utility used during the profiling phase is the elfdump command , which dumps dynamic linking information the object depends on.

[16]URL: http://www.oracle.com/technetwork/server-storage/solarisstudio/overview/index.html

Source Code Analyzer:

The static source code analysis module can check for compatibility issues in, C/C++ source code, K shell and other shell scripts. This module mainly checks against uses of removed/renamed/modified or deprecated Oracle Solaris library functions and shell commands.

RunTime Analyzer:

The runtime analysis module can run on an existing Oracle Solaris 10 test environment, and report the potential compatibility issues. This module of the S11CompatCheckTool uses Oracle Solaris DTrace to look into process details, and may invoke other system tools available on Oracle Solaris 10 (as needed).

The S11CompatCheckTool can be configured to run with different scopes varying from system wide checks to specific processes only.

The Runtime Analyzer tool can report known issues in the following cases

- Invocation of deprecated (EOLed) commands

- Invocation of renamed / relocated commands

- Invocation of commands with deprecated options

- Open / read / write on "deprecated / renamed / relocated" files or devices

- Use of outdated and removed versions of utilities/packages e.g. Hardcoded path to use postgreSQL etc

- Use of removed or replaced device driver interfaces

- Use / invocation of deprecated / EOLed functions / API's / symbols

- Use / invocation of private interfaces

- dlopen of deprecated / renamed libraries

Obtaining the Oracle Solaris 11 Preflight Application Checker

The unbundled version of S11CompatCheckTool can be freely downloaded for X64 and SPARC architectures[17].

Conclusion

Applications correctly developed for Oracle Solaris 10 or earlier Oracle Solaris operating systems are binary compatible with Oracle Solaris 11. The evolution of the operating system in regards of changes to component versions, retired frameworks and command line changes may require changes to make applications run well on Oracle Solaris 11. This documented listed the most likely causes that an application will need to be modified to operate on Oracle Solaris 11.

[17]Download URL: http://www.oracle.com/technetwork/server-storage/solarisstudio/overview/index.html

Oracle Solaris 11 ISV Adoption Guide
May 2012
Authors: Joseph Balenzano, Bruce Chapman, Darryl Gove, Abhishek Gupta, Malcolm Kavalsky, William Knoche, Eric Reid, Stefan Schneider, Caryl Takvorian, Andrew Walton

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

Worldwide Inquiries:
Phone: +1.650.506.7000
Fax: +1.650.506.7200
oracle.com

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

Hardware and Software, Engineered to Work Together