



Developing Applications on the Solaris™ OS and Linux

Frank Liang Lin

January 2007

Sun Microsystems, Inc.

Copyright © 2007 Sun Microsystems, Inc., 4150 Network Circle, Santa Clara, California 95054, U.S.A. All rights reserved.

U.S. Government Rights - Commercial software. Government users are subject to the Sun Microsystems, Inc. standard license agreement and applicable provisions of the FAR and its supplements. Use is subject to license terms. This distribution may include materials developed by third parties.

Parts of the product may be derived from Berkeley BSD systems, licensed from the University of California. UNIX is a registered trademark in the U.S. and in other countries, exclusively licensed through X/Open Company, Ltd. X/Open is a registered trademark of X/Open Company, Ltd.

All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. in the U.S. and other countries. Products bearing SPARC trademarks are based upon architecture developed by Sun Microsystems, Inc.

Sun, Sun Microsystems, the Sun logo, Solaris, Sun Studio, OpenSolaris, NetBeans, Java, Java EE, Java SE, Sun Java Enterprise System, and all Java-based marks are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries.

This product is covered and controlled by U.S. Export Control laws and may be subject to the export or import laws in other countries. Nuclear, missile, chemical biological weapons or nuclear maritime end uses or end users, whether direct or indirect, are strictly prohibited. Export or reexport to countries subject to U.S. embargo or to entities identified on U.S. export exclusion lists, including, but not limited to, the denied persons and specially designated nationals lists is strictly prohibited.

DOCUMENTATION IS PROVIDED "AS IS" AND ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT, ARE DISCLAIMED, EXCEPT TO THE EXTENT THAT SUCH DISCLAIMERS ARE HELD TO BE LEGALLY INVALID.

Table of Contents

Introduction.....	4
Why Develop Applications on the Solaris OS and Linux?	4
Similarities and Differences Between the Solaris OS and Linux.....	4
The Solaris 10 OS Supports Many Open Source Applications.....	5
Common Application Development Issues on the Solaris Platform and Linux.....	6
Open Source Software Libraries.....	6
System APIs.....	7
Multithreaded Programming.....	7
Architecture-Specific Code.....	7
How to Develop Nonnative Applications.....	8
How to Make Existing Linux Applications Run on the Solaris OS.....	9
Porting Applications From Linux to the Solaris OS	12
Application Development Environment on the Solaris OS	12
Build Environment in the Solaris OS.....	13
Packaging Applications and Deploying on the Solaris OS.....	13
More Resources.....	13

Introduction

Operating systems are the platforms for enterprise application systems. Selecting a good operating system for enterprise applications is important for high-quality application systems. This article explains why you should develop applications on the Solaris™ Operating System and Linux, with a discussion of some common application development issues on these two operating systems. Also covered are developing nonnative applications, making existing Linux applications run on the Solaris OS, and porting applications from Linux to the Solaris OS.

Note: In this article, *Linux* refers to GNU/Linux, because Linux itself is just a kernel. When the kernel is combined with the GNU user-level utilities, it becomes an OS.

Why Develop Applications on the Solaris OS and Linux?

In the current professional world of computer software, the Solaris OS and Linux are the most popular UNIX-based operating systems. For application developers, both Solaris and Linux provide rich system commands, programming tools, software packages, and API libraries. They provide good enterprise application development environments. Compared to other operating systems, the Solaris OS and Linux have some system qualities, such as reliability, security, availability, and scalability, in common. These system qualities are very important for enterprise applications, especially for mission-critical enterprise applications.

Because the Solaris OS and Linux are open source-based, developers can use a free distribution method and receive assistance from the open source community, or they can use a subscription-based distribution and receive professional support from a vendor. This flexibility allows developers to choose the model that fits their particular needs. The same choices are also available when end users deploy applications on the Solaris OS or GNU/Linux, allowing for a flexible cost structure.

Similarities and Differences Between the Solaris OS and Linux

The Solaris OS and Linux are both UNIX®-based operating systems, they both implement a UNIX application programming interface (API), that is, POSIX. They both have support for Sun Studio software and GNU tools. Both use X windows as the GUI for the desktop interface. The Solaris OS and Linux are very similar in many ways, just like sons of the same parents. But even twins have differences, and the Solaris OS and Linux also have differences.

The Solaris OS is based on Sys V, and Linux is more of a BSD-style UNIX with some Sys V features. The Solaris OS has always been a core Sun product and technology. Sun regularly invests a large amount of resources, engineers, and funding to research, develop, and add new features. Since the Solaris OS is a core product and Sun has a series of servers, the Solaris OS can run not only on x86/x64 platforms but also on SPARC® platforms. Therefore, if an enterprise application that runs on x86 needs to support more users, you can move the application to a SPARC platform easily. As a core Sun product, the Solaris OS provides compatibility among different versions. If an application runs on the Solaris 2.6 OS, you can easily move it to the Solaris 10 OS with few or no updates, and the application can use many features of the new Solaris version. Sun also provides full support and services for the Solaris OS for Sun customers. Sun has many service plans to meet different requirements for different users. Developers can get Sun support 24 hours a day, 7 days a week. Sun engineers work closely with customers to solve problems on time.

The Solaris 10 OS Supports Many Open Source Applications

The Solaris 10 OS includes and supports many open source applications. Figure 1 is a short list of open source software supported in the Solaris 10 OS. Open source software shown in colored text, such as **Apache**, is fully supported by Sun in the same manner as software developed by Sun.

Figure 1: Open Source Software Supported in the Solaris 10 OS

NETWORK SERVICE & CLIENTS	COMMANDS	LIBRARIES
Apache	a2ps	Glib
Apache2	bzip2	GTK+
bind	footmatic print ppds	JPEG
Mozilla	ghostscript	Libexpat
ncftp	ghostscript fonts	Libusb
PPP	Gimp print drivers	Libxslt
Samba	GNU patch utility	PNG
sendmail	GNU grep less	Tcl/Tk
SER (SIP Proxy Server)	ImageMagick	TIFF
Tomcat	IPMItool	XML2
wget	Open Printing API	XPM
wu-ftpd	rpm2cpio.pl	zlib
xntpd	System Management Agent	Zebra
COMPILERS & TOOLS	SCRIPTING LANGUAGES	SECURITY TOOLS
Binutils	gcc	Secure Shell
Bison	gm4	tcp_wrappers
Flex	gmake	
	Perl	SHELLS
	Python	bash
		tcsh
		zsh

The Solaris OS also bundles many types of open source software. Figure 2 shows a brief list of open source software bundled with the Solaris 10 OS.

Figure 2: Open Source Software Bundled With the Solaris 10 OS

Applications / Accessibility	Applications / Networking	Applications / Utilities		Development / Languages	Development / Libraries
brlty-3.3.1	cups-1.1.20	afio-2.4.6	mpg123-0.59r	bison-1.35	aalib-1.2
emacspeak-18.0	etherreal 0.10.5	amanda-2.4.4	mysql-jdbc-3.0.8	gcc-2.95.3	berkeley-db 1.85
emacspeak-ss-1.9.1	fetchmail 6.2.5	cdrtools-2.01	netpbm-10.3	gcc-3.4.2	berkeley-db 4.2.52NC
freetts-1.1.1	hpijs 1.6	cupsddk 1.0	plotutils-2.4.1	libtool 1.5.2	curl-7.10.3
screenbrlty-4.02	lynx-2.8.4	diffutils-2.8.1	pnm2ppa-1.12	m4-1.4 (GNU)	fitk-1.1.3
unwindows-1.1.3	mutt-1.4.2.1	enscript-1.6.1	rpm-4.1	MySQL python API 0.9.2	fnlib-0.5
w3-4.0.47	nmap 3.5	expect 5.39	sane 1.0.12	php-4.3.2	GD Graphics library 2.0.15
yasr-0.6.4	nmh-1.0.4	file-4.10	screen 4.0.2	ruby-1.6.4	guile-1.3.4
	Open LDAP 2.2.17	fileutils-4.1	sgrep-1.92a	samp-1.0	imlib-1.9.15
Applications / Editors	Open SLP 1.0.11	findutils-4.1.20	sh-utils-2.0	tclX-8.2.0	libxpt-1.95.7
bluefish 0.12	pine-4.61	Foomatic filters 3.0.2	sharutils-4.2.1		libmpeg-1.3.1
emacs 21.3	procmail-3.22	Foomatic-ppds 3.0.1	sudo 1.6.8p5	Development / Tools	libpcap-0.8.3
gawk-3.0.6	rsync 2.6.3 <pre>1</pre>	gcal-3.01	TeX 2.0.2	autoconf 2.59	libsane 1.0.14
joe-3.1	slm-0.9.6.2	gettext-0.10.35	textutils-2.0	automake 1.8.3	linungif-4.1.0
sed-3.02 (GNU)	snort-2.0.0	gimp-print-4.2.6	tnef 1.1.3	binutils-2.15	ncurses-5.2
vim-6.3	topdump-3.8.3	gkrellm 2.1.19	top-3.5.1	cvs 1.11.17	Ogglib-1.0
xemacs-21.4.15		gnuplot 3.7.3	uudeview-0.5.20	ddd 3.3.8	Perl regex lib 4.5
	Applications / Publishing	ispell-3.2.06	vorbis-1.0	gdb 6.2.1	qt-3.1.1
Desktop / Environment	espgs-7.07.1	lxrun 0.9.6.1	wine 20041104	global-4.8	readline-4.2
kde-3.1.1a	graphviz 1.10	mpack-1.5	xpp-1.1	make-3.80 (GNU)	slang-1.4.0
KOffice-1.2.1	groff-1.16.1	mpage-2.5.1			SDL-1.2.5
Xfce-3.8.16	xpdf 3.0				Xaw3d-1.5
	System / Daemons	X / Applications	X / Window Managers		
	imap2002d (UW)	asclock-1.0	afterstep-1.8.8		
	proftpd 1.2.10rc1	etherreal-0.9.11	fvwm2-2.4.3		
	squid 2.5.STABLE7	gimp-1.2.1	WindowMaker-0.80.2		
		rxvt-2.7.10			
		stardic-1.3.1			
		vnc-3.3.7			
		xcpustate-2.5			
		xdelta 1.1.3			
		xmcd 3.2.1			
		xmms 1.2.10			
		xterm-196 (XFree86)			

Common Application Development Issues on the Solaris Platform and Linux

Open Source Software Libraries

There are some common issues when you develop applications on the Solaris platform and Linux. The most common issues are related to open source software libraries. Most applications need open source software support. Many open source software libraries that are used in Linux are already in the Solaris 10 OS and can be used directly, such as `libgtk`, which is in the `/usr/sfw/lib` directory on the Solaris 10 platform. Some open source software is not in the Solaris 10 OS, but you can find a Solaris version from the provider. For example, `zlib` is not currently in the Solaris 10 OS, but you can get the Solaris version at <http://www.zlib.net>. In some cases, you might need to rebuild the libraries from the source code and install them on the Solaris platform.

System APIs

Some system API differences are found between the Solaris OS and Linux, for example, API signatures. For detailed information about API differences, see the *Programming Interfaces Guide* (<http://docs.sun.com/app/docs/doc/817-4415>). Let's just talk about a system API, `fork()`. The API `fork()` creates a new process based on the current process, and the new process is exactly like its parent.

Prior to the Solaris 10 OS, the behavior of the `fork()` API in your application depended on which system library your application linked with. If the application linked with `libthread`, the whole current process was copied to the new process. However, if the application linked with `libpthread`, only the calling thread was copied to the new process. Since creating a new process is resource expensive, the `fork()` API in the Solaris 10 OS copies only the calling thread to the new process. If your application needs to copy the whole process to the new process on the Solaris 10 OS, you can use a new API in this version of Solaris, `forkall()`, to do it.

The API `clone()` in Linux is like `fork()`, but the new process and the parent process share some system resources. The Solaris 10 OS does not support this API and you might need to use another API, such as `fork()`, for your application requirements.

Multithreaded Programming

During application development, you can create multiple threads in a process. These threads share the resources of the process, such as opened files, network connections, or database links; they do different tasks at the same time, and they increase application performance.

Linux threads are heavy-weight and not supported well by tools. Solaris threads are traditional threads; they are light-weight, and they are fully supported by all tools. Both the Solaris OS and Linux support pthreads that follow the POSIX standard. You need to use pthreads for multithreaded programming on both the Solaris OS and Linux, because pthreads let the system libraries handle all the low-level details.

Architecture-Specific Code

When developing applications on the Solaris OS or Linux, you might encounter the issue of architecture-specific code, such as little-endian versus big-endian. There are various descriptions of little-endian and big-endian, which can be confusing to some developers. Little-endian and big-endian issues are related to byte order. If the application data is longer than one byte, the system architecture specifies the byte order in the system that holds the data. As an analogy, when you write a letter to a friend in the United States, you put the street number first, then the street name, and then the city, the state, and the zip code. This order is similar to little-endian. If you write to a friend in Asia, you put the providence first, then the city, and then the street and street number, which is similar to big-endian.

Given the little-endian vs. big-endian issue, you need to convert the data into an architecture-neutral format when transferring data to another computer over a network. There are some APIs, such as `ntohl()`, `ntohs()`, `htonl()`, and `htohs()`, that can be used for the conversions.

Here is an example application that uses a C structure to store the year, month, and day in elements that are of the character data type:

```
struct date {
    char year;
    char month;
    char day;
    char X;
} planned_date, end_date;
```

In the application, you need to control the project schedule. To do this, you need to compare the project deadline date and the real release date:

```
...
if ((* (long *) &end_date) > ((* (long *) &planned_date)) {
    printf("Sorry, the project is late; no bonus for the team this quarter.");
}
...
```

Do you see any problem in the previous code?

Well, the C program assumes the data in the structure will be stored in the computer in the order `year|month|day|X`. But on a SPARC platform, the data will be stored in the order `X|day|month|year`. So, the code runs correctly only on little-endian systems (x86). When you run the code on a SPARC platform, you get different results.

Of course, you like to finish projects on time and also get a bonus, so you need to make sure the code works correctly on different platforms. So, the correct code looks like this:

```
if (cmp_dates(e_date, p_date) > 0)
{
    printf("Sorry, late again!!");
}

int cmp_date(struct d1, struct d2)
{
    if (d1.year != d2.year)
        return(d1.year - d2.year);
    if (d1.month != d2.month)
        return(d1.month - d2.month);
    if (d1.day != d2.day)
        return(d1.day - d2.day);

    return 0;
}
```

How to Develop Nonnative Applications

You might think it is too complicated to handle the previous issues when developing applications on the Solaris OS and Linux, and you might wonder if there is any way you can develop applications on both OSs without facing these issues. Yes, there are technologies to develop nonnative applications.

One of most common ways to make applications portable is to use Java™ technology, which includes Java Standard Edition (Java SE), Java Enterprise Edition (Java EE), and many other technologies. You

can use Java technology to develop applications easily on one platform and deploy them on another platform without any change.

For enterprise applications, you can use technologies such as JavaServer Pages™ (JSP™), servlets, or Swing to create a GUI; POJO or Enterprise JavaBeans™ (EJB™) to process data according to business rules; and Java Database Connectivity™ (JDBC™) to save data into databases. Java technology offers many features and functions for a wide range of business applications.

The Solaris Enterprise System includes the Java SE platform, and you can use it as soon as you install the Solaris 10 OS. The Solaris 10 OS also encompasses the Sun Java Enterprise System, which includes containers for the Java EE platform. The Solaris 10 OS supports most common Java EE containers such as Tomcat, Apache, JBoss, and so on.

Other technologies you can use to develop nonnative applications include PHP, Perl, Ruby, and so on. These technologies are available on the Solaris OS. Using these technologies, you can run applications on both the Solaris OS and Linux. Sun's an open source software package named Cool Stack is optimized for this purpose and is available at <http://cooltools.sunsource.net/index.html>.

Cool Stack contains the following software:

- Package CSQamp:
 - Apache HTTP Server 2.0.58
 - MySQL 5.0.22
 - PHP 5.1.4
 - MySQL (32-bit version)
- Package CSQmysql (64-bit version of MySQL 5.0.22 with innodb)
- Package CSQperl (Perl 5.8. optimized with Sun Studio software)
- Package CSQphp (PHP 5.1.4)
- Package CSQsquid (Squid 2.5.STABLE14)
- Package CSQtomcat (5.5.17)

With Cool Stack, you can use Apache or Tomcat to create a GUI, use Perl or PHP to process data according the business rules, and use MySQL to save the data into databases.

How to Make Existing Linux Applications Run on the Solaris OS

If you have read until this point, you might think it is great to use these technologies to develop new applications on the Solaris OS, but you might wonder whether you can run existing Linux applications on the Solaris OS.

The answer is yes. If you have existing Linux applications and you want to run them on the Solaris OS, there are several ways to do it.

The most common way is porting. You need to set up the configuration of the application, rebuild your application, and deploy it on the Solaris environment. Many Linux applications can be ported easily into the Solaris OS.

You might prefer to run existing Linux applications on the Solaris OS without porting. Sun has a feature called Solaris Containers for Linux Applications (SCLA), which is also called BrandZ. Linux applications can run in BrandZ without any change.

Before learning about BrandZ, let's talk a bit about zone technology. There are some common problems in the real world for enterprise applications. In a product environment, if an application has a problem and brings the system down, it affects other healthy applications that are running on the same server. In another case, if a hacker breaks into an application on a server, the hacker can use the privileges of that broken application to attack other healthy applications on the same server. Solaris Zones, part of Solaris Containers technology in the Solaris 10 OS, provide a solution for these problems. Zones are OS instances for user applications. An application that runs on a zone is isolated from other applications that run on other zones, so the application does not affect other applications in different zones.

BrandZ is a special zone created for Linux applications running on the Solaris 10 OS. BrandZ runs Linux `init(1M)`, and it runs Linux configuration scripts for applications. But it is run on the top of the Solaris kernel instead of on Linux. In BrandZ, there are no Linux binaries or packages, and BrandZ is not a Linux distribution. BrandZ does not include Linux software. BrandZ is based on selected, standard Linux distributions that initially came with Red Hat AS 3 and equivalent CentOS release. So, based on this, you cannot run any random Linux distribution or other OSs on BrandZ. No Linux kernel is ever executed on BrandZ and BrandZ does not support all Linux functionality, such as Linux file systems or third-party kernel modules.

If you are interested in BrandZ, you can find out more about the BrandZ community at <http://www.opensolaris.org/os/community/brandz/>.

Following is an example that shows how to start BrandZ in the Solaris 10 OS and how to use DTrace to monitor applications that are running in BrandZ.

In this example, you first determine that the OS is a Solaris OS by using the `uname` command. Then you start a predefined BrandZ zone called `linux` by using the `zoneadm -z linux boot` command. After you log in to the `linux` zone, you use the `uname` command again and it shows the OS is now Linux.

```
bash-3.00# uname -a
SunOS DaLianBoston 5.11 brandz-builds_2006-08-17 i86pc i386 i86pc
bash-3.00#
bash-3.00# zoneadm -z linux boot
bash-3.00# zlogin linux
[Connected to zone 'linux' pts/5]
Welcome to your shiny new Linux zone.
```

- The root password is 'root'. Please change it immediately.
- To enable networking goodness, see `/etc/sysconfig/network.example`.

- This message is in /etc/motd. Feel free to change it.

For anything more complicated, see:

<http://opensolaris.org/os/community/brandz/>

You have mail.

```
-bash-2.05b#
-bash-2.05b#
-bash-2.05b# uname -a
Linux linux 2.4.21 BrandZ fake linux i686 athlon i386 GNU/Linux
-bash-2.05b#
-bash-2.05b#
-bash-2.05b# ls
java local netbeans runstudio SUNWspro test.dat t.tar
-bash-2.05b# more test.dat
this is a test linux brandz
-bash-2.05b#
```

In the BrandZ example, when you start the BrandZ linux zone, you can run a DTrace script at a global zone.

DTrace is a dynamic tracing facility of the Solaris 10 OS. When you develop an application, you use a debugger to find and fix problems in the programs. But debuggers have some limitations. When you debug system API calls, you cannot continue to dig deeper, so you do not know what is going on inside the system calls. It is difficult to solve application performance problems using a debugger, and debuggers can do almost nothing about conflicts between applications that are running on the same server. Most importantly, debuggers cannot be used in production environments. DTrace provides many functions that provide "inside information" about operating systems and system calls. DTrace has a D script language and you can use it to write your own DTrace scripts to trace special processes and functions. DTrace is a light-weight facility, and you can dynamically turn it on and off, so you can use it in production environments. To learn more about DTrace, see the *DTrace User Guide* at <http://docs.sun.com/app/docs/doc/819-5488>.

Back to the example: In a global zone, the output of the DTrace program is as follows. **Note:** The notes within <...> are comments, not the output of DTrace scripts.

```
bash-3.00# ./execsnoop
  UID   PID   PPID  ARGS
<trace "uname" command>
  0    2098   1525  uname -a
<trace "zoneadm" command in another terminal>
  0    2099   1525  zoneadm -z linux boot
  0    2100   2099  zoneadmd -z linux
  0    2103   2101  mount -o zonedevfs /export/home/zones/linux/dev
/export/home/zones/linux/root/d
  0    2104   2101  mount -o zonedevfs /export/home/zones/linux/dev
/export/home/zones/linux/root/n
  0    2105   2101  mount -o ro /lib /export/home/zones/linux/root/native/lib
  0    2106   2101  mount -o ro /usr/lib /export/home/zones/linux/root/native/usr/lib
  0    2107   2101  devfsadm -z linux
  0    2116   2101  sh -c exec /usr/lib/brand/lx/lx_boot /export/home/zones/linux linux
  0    2116   2101  /bin/sh /usr/lib/brand/lx/lx_boot /export/home/zones/linux linux
  0    2117   2116  cp -p
/var/sadm/pkg/SUNWnfscr/save/pspool/SUNWnfscr/reloc/etc/default/nfs /expo
  0    2118   2116  rm -f /export/home/zones/linux/root/.autofsck
  0    2119   2116  /usr/lib/brand/lx/lx_audio_cfg boot linux
  0    2120   2102  /sbin/init
```

```

0 2123 2122 /bin/bash /etc/rc.d/rc.sysinit
0 2123 2122 /sbin/initlog -r /etc/rc.d/rc.sysinit
<skip some detailed DTrace output>
.....

<trace "zlogin" command>
0 2407 1525 zlogin linux
0 2409 2408 pt_chmod 3
0 2408 2407 /usr/bin/login -f root
0 2411 2408 -bash
0 2413 2412 id -u
<skip some detailed DTrace output>
....

<trace "uname" from BrandZ linux console>
0 2508 2411 uname -a
<trace "ls" from BrandZ linux console>
0 2509 2411 ls -color=tty
<trace "more" from BrandZ linux console>
0 2510 2411 more test.dat

```

So, you can do performance tuning on Linux programs that run on BrandZ by using DTrace scripts that run in the global zone on the same server.

For more information on DTrace, see:

- "Using DTrace to Profile and Debug a C++ Program"
http://developers.sun.com/solaris/articles/dtrace_cc.html
- "DTrace Case Study for Developers"
http://developers.sun.com/solaris/articles/dtrace_for_dev.html
- "Using DTrace with Sun Studio Tools to Understand, Analyze, Debug, and Enhance Complex Applications"
<http://developers.sun.com/sunstudio/articles/dtrace.html>

Porting Applications From Linux to the Solaris OS

Application Development Environment on the Solaris OS

Sun Studio software offers the application development environment for the Solaris OS. The Sun Studio 11 release is a complete set of record-setting, optimizing compilers and tools for C, C++, and Fortran. Sun Studio software is available on the Solaris OS, and it is also available for Linux. If you use Sun Studio tools to develop applications, it is easy to port your applications from Linux to the Solaris platform. Like most integrated development environments (IDEs), Sun Studio 11 tools are easy to use and developer-friendly. For more information, see <http://developers.sun.com/sunstudio>.

You can get the most up-to-date version of Sun Studio at <http://developers.sun.com/sunstudio/downloads/express.jsp>.

If you use the GNU family of compilers on Linux and you would also like to use the GNU compiler for the SPARC platform, it is available for the Solaris OS at <http://cooltools.sunsource.net/gcc/>.

Build Environment in the Solaris OS

After you design and write your code on the Solaris platform, you need to create or "make" the application. The Solaris make command and the GNU make command are similar. Both define the same basic set of suffixes and implicit rules, such as the same variables, internal macros, and targets.

If you would like to use the GNU make command on the Solaris platform, you can get it at <http://www.sunfreeware.com>.

Packaging Applications and Deploying on the Solaris OS

Applications on the Solaris OS use the Sys V package as the default package format. Linux packages use the RPM format. The RPM format is found at <http://sunrpms1.maraudingpirates.org:8080/>, if you want to use it on the Solaris OS. Be aware that RPM does not understand Solaris Containers.

If you package your Linux applications on the Solaris OS and deploy them on the Solaris OS, and you want to convert RPM packages to Solaris packages, you can get the make_pack package at <http://www.sunfreeware.com/pkgadd.html>.

The following commands can be a good guide for how to build packages from RPM:

```
bash# mkdir -p $HOME/tmp/myprogram-x.y/usr/local
bash# cd $HOME/tmp/myprogram-x.y/usr/local
bash# rpm2cpio rpm-4.0.2-8.X64.rpm | cpio -dimv
bash# tar -xvf rpm*.tar.gz
bash# make_pack
```

More Resources

The Solaris Developer Center offers resources such as:

- Information about application development on the Solaris platform: <http://developers.sun.com/solaris/>
- Training information: <http://developers.sun.com/solaris/learning/training.jsp>
- White papers: <http://developers.sun.com/solaris/reference/docs/whitepapers.jsp>
- Linux Compatibility Assurance Toolkit (LinCAT): <http://developers.sun.com/solaris/downloads/lincat/>

And here are more resources for learning about the Solaris OS and related technologies:

- Get the Solaris OS: <http://sun.com/solaris/get>
- Get started with the Solaris OS "how to" guides: http://www.sun.com/software/solaris/howto_guides.jsp
- Get information about the CoolTools community: <http://cooltools.sunsource.net>
- Get freeware for the Solaris OS: <http://www.sunfreeware.com/>

Licensing Information

Unless otherwise specified, the use of this software is authorized pursuant to the terms of the license found at http://developers.sun.com/berkeley_license.html.