

Inside Solaris™

Tips & Techniques for users of Sun Solaris

Using Apache as a proxy server

by Don Kuenz

More than half of the Web sites on the Internet use a free software package named Apache as their Web page server. Did you know that you can also use Apache as a proxy server? In this article, we'll show you how to do so by taking a closer look at how Apache proxy fits into a network and by creating a simple network that contains two Windows 98 client hosts

and one Solaris Apache proxy, as shown in **Figure A**.

Understanding what Apache proxy does

Before you install Apache proxy, let's spend a little time discussing what this particular proxy does. In the parlance of computer networks, the word proxy means many different things. In the context of this article, *proxy* means a process that sends and receives data on behalf of an Internet browser such as Netscape or Internet Explorer (IE). Apache only proxies the following services: http, https, ftp, socks, gopher, and WAIS. You need to find another method, however, to proxy services such as telnet, DNS, and SMTP.

Many people use Apache proxy to allow a private network to share a public IP address. Typically, at the low end of the market, your Internet Service Provider (ISP) only provides one public IP address. If you happen to use a private network at your site, this means that only one host at a time can use the public IP address, which means that only one person at a time can access the Internet through your ISP. Apache proxy allows you to effectively share a single public IP address by providing simultaneous Internet Web page access to all hosts. That way, multiple people can access the Internet at the same time.

Many people also use Apache proxy to insulate their own network from the Internet, with its attendant hacker element. For the sake of security, you want to make it hard for a hacker to see your network from the Internet.

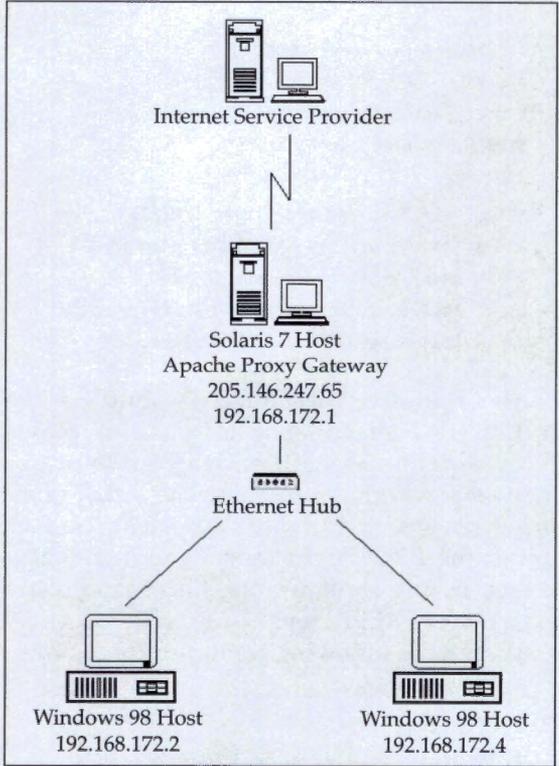


Figure A: This diagram shows a simple network that uses Apache proxy.

Securing your networked systems with Solaris 7

Introducing Message Digest algorithm, version 5

Packaging in Solaris

Solaris moves towards open source

Code coverage and profiling with Tcov

Solaris Q & A:

- Making cron jobs quiet
- It's not a bug, it's a feature
- Network management for free
- Solaris Device Configuration Assistant

Dual booting Solaris and Linux

This means that you probably should run Apache proxy even if you enjoy the luxury of hundreds of public IP addresses at your disposal. You especially need to use it to protect hosts that run Windows 98, which suffers from a legendary lack of security.

Apache and Sun

The low cost of Apache software, coupled with the great bargains currently available on reconditioned Sun hardware, opens a tremendous marketing opportunity for Sun products among cost-sensitive small businesses. This allows Sun to effectively compete in the small business market—a market that traditionally has belonged to Microsoft. This market will start purchasing proxy solutions in the near future, and cost is always a big factor in any decision. The high-cost of Microsoft's proprietary proxy server makes it cheaper to purchase a reconditioned Sun running Solaris with Apache proxy. Now that we understand how Apache proxy works and appreciate some of the benefits of using it, let's take a closer look at how Apache proxy fits into a network.

A proxy network

To take a closer look at Apache proxy, let's create a simple network that contains two Windows 98 client hosts and one Solaris Apache proxy. The Solaris host connects to the Internet through an ISP. **Figure A** shows our simple network.

Notice that our network assigns three private IP addresses (192.168.172.1, 192.168.172.2, and 192.168.172.4) to the hosts connected to our private network. As noted in RFC1918, public Internet routers drop packets destined for IP addresses, which start with 192.168. However, we can build private routes within our own network to assign these very same private IP addresses to hosts under our direct control.

The lack of a public route to our private addresses keeps hackers from directly attacking our Windows 98 hosts. Unfortunately, it also prevents Web page packets from finding their way back to our Windows 98 hosts. Our Apache proxy host adds the missing functionality. Web page packets can find their way back to it using its public IP address. It also knows about our private routes, which enables it to send and receive Web pages on behalf of our Windows 98 hosts.

You can also see that our Solaris host uses one public IP address (205.146.247.65) and one private IP address (192.168.172.1). The public address enables hackers to directly attack that host. You'll definitely want to increase the security of your Apache proxy host using the techniques covered in previous issues of *Inside Solaris*.

Installing Apache

You must compile a proxy module (mod_proxy) into Apache before you can use it as a proxy server. Most of the precompiled binaries available on the Internet leave it out, because Apache's default configuration disables mod_proxy. This means that you need to obtain, configure, and compile the source.

You can obtain the latest version of the source at www.apache.org/dist/. This article uses version 1.3.9 (the latest version at the time of this writing). You also need an ANSI C compiler from either Sun or GNU.

After you unpack your source, take a look at a file named INSTALL that lives in the top-most source directory. INSTALL contains the following overview for the impatient:

1. Overview for the impatient

```
-----  
$ ./configure --prefix=PREFIX  
$ make  
$ make install  
$ PREFIX/bin/apachectl start
```

NOTE: PREFIX is not the string "PREFIX".
Instead use the UNIX filesystem path
under which Apache should be
installed. For instance use "/usr/
local/apache" for PREFIX above.

You'll probably like Apache's simple, clean install. If everything goes well, it plows through its build with only a couple of unavoidable warning messages. The install minimizes file system pollution by placing all files under the PREFIX directory. The install also preserves any configuration files that already exist under PREFIX/etc.

We use the following configure command:

```
CC="cc" ./configure \  
--prefix=/usr/local/apache \  
--enable-module=proxy \  
--enable-shared=proxy
```

Notice that we stick with current conventional wisdom and place Apache files under the /usr/local/apache directory. You may use another directory by changing

```
--prefix=/usr/local/apache
```

We also use Sun's compiler. If you prefer to use GNU's compiler, drop CC="cc" from the front of configure.

After configure finishes, type *make* at the command prompt to build everything. Then, log on as root and type *make install*. Finally, type `/usr/local/apache/bin/apachectl start` to start Apache. You can use Netscape to verify your installation. Open your Apache host's name as a URL. Netscape will now display an Apache welcome page or your server's original home page (if one exists).

Configuring proxy

After you verify that your Apache correctly runs as a standalone Web server, you need to configure your network to use it as a proxy server. This involves changing a configuration file named `httpd.conf`, which resides on the Apache host. You also must alter Netscape's proxy configuration on each client host.

You can find the `httpd.conf` file, which you must change, in the `/usr/local/apache/conf` directory. You need to change two lines in this file. Open the file with your favorite editor and search for a line that contains:

```
# LoadModule foo_module libexec/mod_foo.so
```

Under that line, add another line:

```
LoadModule proxy_module libexec/libproxy.so
```

Next, search for, and uncomment, the following lines:

```
<IfModule mod_proxy.c>  
ProxyRequests On
```

Finally, save `httpd.conf` and force Apache to use the new configuration by stopping it, and then restarting it with the following commands:

```
/usr/local/apache/bin/apachectl stop  
/usr/local/apache/bin/apachectl start
```

Your Apache can now act as a proxy server.

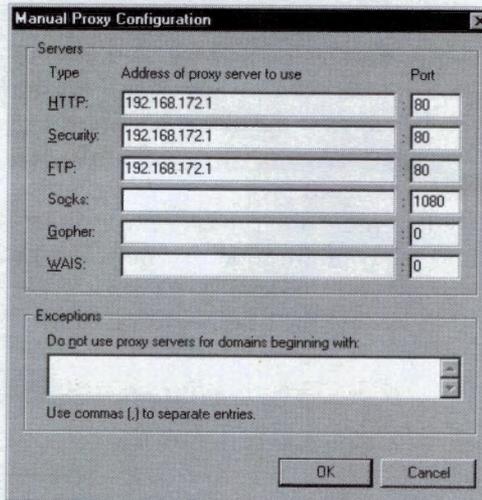


Figure B: This is Netscape's Manual Proxy Configuration dialog box.

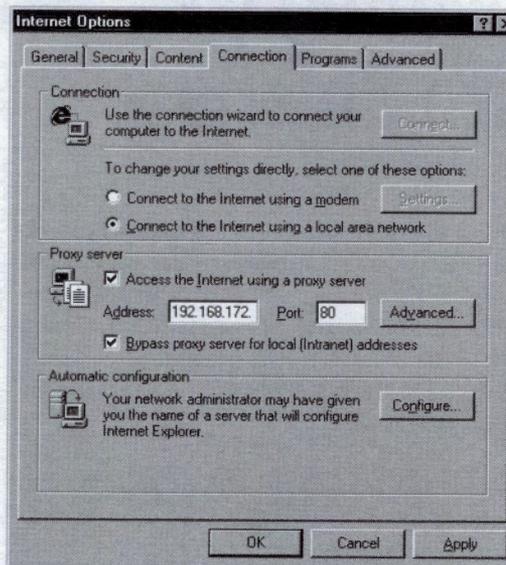


Figure C: This is Microsoft's Internet Explorer Proxy server dialog box.

Now, you only need to configure the browser on each of your client hosts, and you can start using Apache proxy server. Figure B shows the appropriate Netscape Communicator settings for our Apache proxy server, which uses a private IP address of 192.168.172.1. You find this dialog box under `Edit | Preferences | Advanced | Proxies`. Select the Manual Proxy Configuration option button, and then click the View button. You should now see the dialog box shown in Figure B.

Microsoft's Internet Explorer also contains a proxy configuration dialog box. You find it under View | Internet Options. Click on the Connection tab, to view the dialog box shown in **Figure C** on page 3. Again, we use 192.168.172.1 as the IP address of our proxy server.

After you enter the proxy server information into the configuration dialog box, click OK to apply your changes. At this point you can start surfing the Internet using Apache as your proxy.

Conclusion

Apache can function as a proxy server to allow multiple client hosts to access Internet Web pages through a single shared public IP address. This increases security because it insulates clients from direct Internet access. In order to enable Apache's proxy functionality, you must obtain its source code, build it, install it, and configure it. Then you need to use your browser's proxy configuration panel to activate it on the clients. 

Securing your networked systems with Solaris 7

by Edgar Danielyan

The ultimate goal of computer networks is to provide convenient access to computing power, data, and applications distributed across the network. At the same time, it's also necessary to provide these services only to authorized users and only for authorized use.

Meeting both of these needs can be difficult, and raises the question, "How can I provide convenient access, but at the same time restrict it to authorized users only?" There are many approaches to networked systems security; all of them have their pros and cons. The best approach, however, is to balance two requirements—provide access without compromising security and have a secure system without making it too inconvenient to use.

In this article, we'll look at the security features offered by Solaris 7 and give some recommendations on how to make the best use of them, as well as briefly describe freely available security software. Knowledge of available security features coupled with common sense will minimize the risk of potential unauthorized use, modification, or damage to networked computer systems, but not eliminate it altogether. No doubt, with the advancement of distributed computing and e-commerce, security risks won't only increase but will also cre-

ate different security concerns than the centralized mainframe era. We have to keep up with the new security threats and have the necessary security in place, on time.

Solaris 7 security features

There are many specialized security solutions available for Solaris, both from Sun Microsystems and other vendors. High-quality software is also available on the Internet for free.

It's necessary to note that a system is as strong as its weakest link; therefore, there's no point in defining a tight security policy when, for example, the root's password may be easily obtained by eavesdropping on a careless administrator's telnet connection.

Passwords

The oldest and most conventional way of access control is passwords. While not the best solution for access control available, passwords continue to remain the most widespread access control concept. Password security is comprised of two interrelated parts: the technical security and social security. That is, you may have an ideally configured system, but a password written down on a piece of paper and then thrown away may end up in a potential intruder's hands with clearly pre-

dictable results. On the technical side, Solaris 7 provides the following password and account security features:

- Passwords are never stored in plain text—that is, if someone gets your `/etc/shadow` file, it doesn't mean they have the passwords.
- A feature called password aging provides for some sort of control over the users' use of passwords by forcing them to change passwords as frequently as deemed necessary.
- When changing passwords, users aren't allowed to use the same password again—the older your password gets, the higher the risk that it isn't secret anymore.
- Passwords are checked to make sure they aren't too easy to guess—they must contain a combination of letters, numbers, and other symbols.
- An account expiration feature allows administrators to set a lifetime period for particular accounts, thus eliminating the possibility of unauthorized use outside of the set timeframe.
- The default behavior for the root account is to allow access only from the console and `su` attempts may be logged on several devices (in addition to normal logging).

Note that these particular features deal with authentication issues only and clearly aren't enough by themselves. An important consideration in defining a system's security policy is to collect and keep system logs. The best approach would be to archive all logs, say once a month, on a CD-ROM, so if you should later need to check what happened on a particular date, you would have all the required information. You can configure these settings in `/etc/default/passwd` and `/etc/default/login`.

File Access Control Lists

Solaris provides POSIX 1003.6 compliant access control lists (ACLs) on both UFS and NFS file systems. These increase control over access to particular files/directories by fine-tuning their access permissions.

Auditing

Solaris has complete support for auditing, in addition to standard UNIX logging. In case of

a violation or attack attempt, an audit trail may be of enormous help in the process of discovering what happened and what (or who!) is to blame. However, auditing is a heavyweight process and you should use it with care—on low-end systems, auditing may take as much resources as the processes being audited.

R commands

Commands like `rlogin`, `rexec`, and `rcp` are very vulnerable to various risks; access control methods used by them are outdated and many successful attack strategies are in place to fool them. Therefore, they shouldn't be enabled on mission-critical systems. Not only do they transmit sensitive information such as passwords in clear text over the network, where this information may be easily captured, they also rely on external, non-secure services (such as DNS) for authentication purposes.

Automated Security Enhancement Tool

Solaris 7 includes the Automated Security Enhancement Tool (ASET). This tool helps system administrators set security levels and track changes in the system that may affect security. Our February article, "Securing systems with ASET," by Alan Orndorff, provides a good overview of ASET's functionality.

The minimalist approach

An old Armenian proverb says, "If there are no doors no one would be able to break them." Look at your `/etc/inetd.conf` file, which is one of the most vulnerable configuration files. By default, there are many services configured that aren't used on all machines.

You definitely don't need a finger on a single user workstation or the talk daemon on a machine that has only a root account. Get rid of all unused services by commenting them out.

Also, pay attention to small servers, such as echo, daytime, and others. Leave in only the necessary ones. Take this approach to all other configuration files, as well—the less doors you have, the less likely they are to be broken into.

The heavy arms approach

Another approach, which is much more aggressive than the minimalist approach described previously, is to use all tools available to prevent the breach of security, or, if the

intruder is able to get in, trace him and have enough information to seek legal recourse. The list of freely available software in **Table A** may greatly help achieve these goals.

Of course, this list isn't exhaustive. System security is dynamic—security holes are discovered and tried every day, and it's impossible to overestimate the importance of keeping up with the latest developments. One of the best ways to do this is to subscribe to CERT Advisories, a mailing list provided by the Computer Emergency Response Team (CERT) Coordination Center (www.cert.org) at the Software Engineering Institute of the Carnegie Mellon

University. For information on how to subscribe to this list, see www.cert.org/contact_cert/certmaillist.html.

Obtaining and installing Solaris patches from Sun Microsystems is also a mandatory security requirement. These patches are available for free at sunsolve.sun.com.

Summary

Having secure software doesn't mean having a secure system. The software is merely a tool to help accomplish a certain goal—the accomplishment depending equally on the tool and the system administrator. 

Table A: Freely available software

Software	Location	Description
SSH	ftp://ftp.cs.hut.fi/pub/ssh	SSH (Secure Shell) is a plug-in replacement for rsh, telnet, rlogin, and rcp, providing encryption, authentication, and compression in one easy-to-use and very secure program.
PGP	web.mit.edu/network/pgp.html	PGP (Pretty Good Privacy) is a powerful, cryptographic software suite that enables you to securely exchange messages and to secure files, disk volumes, and network connections with both privacy and strong authentication.
cops	ftp://ftp.cert.org/pub/tools/cops	This is a set of programs, each checking a different aspect of security on a UNIX system. If any potential security holes do exist, the results are either mailed or saved to a report file.
crack	ftp://ftp.cert.org/pub/tools/crack	A program designed to find standard, UNIX, eight-character, DES encrypted passwords by standard guessing techniques.
Deslogin	ftp://ftp.uu.net/pub/security/des	A remote login program that you can use safely across insecure networks.
Gabriel	www.lat.com/	A SATAN detector. Gabriel gives the system administrator an early warning of possible network intrusions by detecting and identifying SATAN's network probing.
opie	ftp://ftp.sunet.se/pub/security/tools/password/nrl-opie/	Provides a one-time password system for POSIX-compliant, UNIX-like operating systems.
SATAN	http://www.fish.com/satan	The Security Analysis Tool for Auditing Networks (SATAN). In its simplest (and default) mode, it gathers as much information about remote hosts and networks as possible by examining such network services as finger, NFS, NIS, ftp and tftp, rexd, and other services.
tcpwrap	ftp://ftp.win.tue.nl/pub/security/	Monitors, logs, and controls remote access to your local tftp, exec, ftp, rsh, telnet, rlogin, finger, and systat daemons.
tripwire	ftp://ftp.cert.org/pub/tools/tripwire	Monitors the system and logs break-in attempts.

Introducing Message Digest algorithm, version 5

by Edgar Danielyan

Message Digest algorithm, version 5, or MD5, as it's known, is widely used in modern computing in general and in the Solaris operating environment in particular. In this article, we'll tell you all about MD5 as well as provide you with a brief description of this popular algorithm.

Background

MD5 was invented by Ronald Rivest, who was at the time working at the Laboratory for Computer Science of the Massachusetts Institute of Technology. Then, in April 1992, MD5 was published as RFC 1321. Later on, Rivest, with Shamir and Adleman, found RSA Data Security, one of the most well-known and respected names in the cryptography and security industry. Such widely used protocols as Hypertext Transfer Protocol Secure (HTTPS) and Secure Sockets Layer (SSL) use RSA algorithms and software.

The algorithm

MD5 takes a bit pattern of arbitrary but finite length and produces a 128-bit digest of that pattern. Note that regardless of the length of the bit pattern, the digest is always 128 bits long. It's very difficult to produce two patterns having the same digest or to produce a message having a predetermined digest.

The algorithm itself isn't very difficult and doesn't require big substitution tables; version 5 of the algorithm is specifically optimized for 32-bit processors. The previous version (MD4) is a little bit faster than MD5, but is less secure and isn't widely used. Security experts estimate that the difficulty of finding two-bit patterns having the same digest is 2^{64} operations and the difficulty of finding a bit pattern having a predetermined digest is 2^{128} operations.

Being in the public domain, MD5 is available to any user without any restrictions, and is being used in almost all applications requiring digital signatures or a way to obtain a hash of a particular bit pattern to be used as a checksum. An incomplete, but representative list of software systems that use MD5 includes: Solaris Operating Environment, Sun WebServer (Sun Microsystems), Netscape Communicator, Netscape Web servers (Netscape Communications), and Cisco IOS (Cisco Systems). RFC 1321, where MD5 is described, also contains a reference implementation of MD5 algorithm in ANSI C that works on all platforms with ANSI C compiler (including Solaris on SPARC and x86). 

References

- Rivest, R. (1991) "The MD4 message digest algorithm," in A.J. Menezes and S.A. Vanstone, (Editors) *Advances in Cryptology: Crypto, 90: Proceedings (Lecture Notes in Computer Science, Vol. 537)*.
- Rivest, R. (April 1992) "The MD4 Message Digest Algorithm," RFC 1320, MIT and RSA Data Security, Inc.
- RSA Security Inc.
www.rsasecurity.com
- RSA Laboratories
www.rsasecurity.com/rsalabs/
- Laboratory for Computer Science at Massachusetts Institute of Technology
www.lcs.mit.edu

ZD Journals

MORE?
Looking for

Visit our new pay-per-view resource library at www.zdjournals.com/get/ejournals for access to over 20,000 articles from 26 different ZD Journals publications.

Packaging in Solaris

DOWNLOAD
ftp.zdjournal.com/sun/mar00

by Hariharan S

Most software that's available for the Solaris platform is delivered using the pkgadd format. This format provides a convenient way for developers to distribute their software to customers and users. In this article, we'll show you how to create your own package in Solaris. It's not a difficult job, but it requires that you follow the proper procedures. Let's see how to create a simple package.

What's in a package?

A Solaris package consists of a series of files that describe the software being delivered. At a minimum, a package requires the pkginfo and prototype files.

A *pkginfo* file is an ASCII file that lists the characteristics of a package. It contains information like name, architecture, version, vendor etc. We can also include our own parameters.

A *prototype* file is also a text file. Each entry in this file describes a package object. This has information like file type, path, permissions, owner, group etc.

Information files

You can also use the *depend*, *copyright*, and *space* files in your package to improve its functionality. *Depend* declares the dependencies of the package and has the following format:

Table A: Scripts for checking the conditions and for proper installation of the package

Script	Use
Request	Obtains input from the installer
Checkinstall	Checks system data
Preinstall	Performs any custom installation needs before installation
Postinstall	Executes after installation
Preremove	Checks some conditions before removal
Postremove	Cleans up after removal
Class action	Executes for each class of objects

Type pkg name

Arch version

Arch version

where Type can be P for prerequisite, I for incompatible, or R for reverse compatibility.

The *Copyright* file contains vendor copyright information. For example, Sun could have a line in their copyright file as follows:

```
Copyright 1999 Sun Microsystems, Inc. All rights reserved.
```

The *space* file is used for reserving additional space in the target system. Space has the following format:

```
pathname blocks inodes
```

For example,

```
/opt 500 50
```

will reserve 500 blocks and 50 inodes in the /opt file system.

Installation scripts

We can also use a set of scripts to ensure the system is in the proper condition for installation of the package. Take a look at **Table A** for a list of scripts.

Creating a sample package

To begin creating our sample package, which we'll call ZDJtest, we have to collect all the objects that go with our package. Assume the following files make up our package:

- Executables—zджtest
- Libraries—libzджtest.so
- Data file—zджtest.dat
- HTML file—zджtest.html

We can arrange these in source directory srcdir as:

```
$ cd srcdir  
$ ls -R .
```

```
./bin:
zdjstatus
./etc:
zdjtest.dat  zdjtest.html
./lib:
libzdjtest.so
```

Next, create the pkginfo file shown in **Listing A**. You'll find that **Listing A** is self-explanatory.

Next you have to create the prototype file. You can do so using pkgproto command. Go to srcdir and execute the following commands:

```
$ cd srcdir
$ pkgproto . > prototype
```

This output from pkgproto isn't complete. We have to do some customization. The output from pkgproto is:

```
d none bin 0755 hharan staff
f none bin/zdjtest 0755 hharan staff
d none lib 0755 hharan staff
f none lib/libzdjtest.so 0644 hharan staff
d none etc 0755 hharan staff
f none etc/zdjtest.html 0644 hharan staff
f none etc/zdjtest.dat 0644 hharan staff
```

Information files

The copyright file contains copyright information such as

Listing A: Our example Pkginfo file

```
#PKG gives abbreviation for the package
PKG=ZDJtest
# Name of the package
NAME= test package
VERSION=1.0
ARCH=i486
# Category can be system, application or user
defined
CATEGORY=application
#the above fields are mandatory
EMAIL=hharan@mailcity.com
VENDOR= ZD journals
# This is our own parameter
SPACE_REQUIRED=4000
# It is conventional to store packages in /opt
BASEDIR=/opt/ZDJtest
```

```
Copyright (c) 2000 Test inc all rights
reserved.
```

The depend file contains references to all the software that our package depends on. Suppose we use tcl-tk to execute our software. Because this package is a prerequisite for us, we will put the following in our depends file:

```
P TCL TCL-TK software
```

Installation scripts

These scripts are optional. Our example uses the scripts shown in **Listing B**. These scripts

Listing B: Installation scripts that complete the package

```
Request file:
PATH=$PATH:/usr/sbin:/usr/sadm/bin
if valuid testuser
then
  if valgid testgrp
  then
    exit 0;
  else
    echo "group testgrp doesn't exist";
  fi
else
  echo "user testuser doesn't exist";
fi
echo "Do you want to continue ? ";
read ANSWER;
if [ "$ANSWER" = "n" -o "$ANSWER" = "no" ]
then
  echo " Processing terminated at user's request."
  exit 1
fi
exit 0;

checkinstall:
# This is available only from Solaris 2.5.
# Use preinstall for earlier versions.
```

```
SPACE_AVAIL="" /usr/bin/df -k /opt | \
sed -e 's/ */ /g' | egrep -v 'avail' | cut -d' ' -f 4"
if [ $SPACE_AVAIL -lt $SPACE_REQUIRED ]
then
  echo "Space not available ! "
  exit 1;
fi
# We can also create testuser and testgrp here
# if they don't exist
exit 0;

postinstall:
# We can create additional files such as links
# if required.
if [ ! -f /bin/zdjtest ]
then
  /usr/bin/ln -s /opt/ZDJtest/bin/zdjtest /bin/zdjtest
fi
postremove:
# We can remove temporary files created
# by postinstall here.
if [ -f /bin/zdjtest ]
then
  /usr/bin/rm /bin/zdjtest
fiR
```

execute with super user permissions in the target system. Paths of commands used in the scripts should be proper.

Customizing the prototype file

You have to create entries for the information scripts in the prototype file. These will have a file type of `i`. You should also modify owner and group entries as per the requirements. The final prototype file is given in [Listing C](#).

Pkgmk command

Now, copy all information and installation scripts to the `basedir`. Then, run the `pkgmk` command to build an installable package:

```
$ cd srcdir
$ pkgmk -b srcdir -d destdir
```

This command creates a package in `destdir`. We should give a full path for `basedir`. Let's see

Listing C: *The prototype file after customization*

```
# prototype file
i pkginfo
i request
i checkinstall
i postinstall
i postremove
i depend
i copyright
d none bin 0755 testuser testgrp
f none bin/zdctest 0755 testuser testgrp
d none lib 0755 testuser testgrp
f none lib/libzdctest.so 0644 testuser testgrp
d none etc 0755 testuser testgrp
f none etc/zdctest.html 0644 testuser testgrp
f none etc/zdctest.dat 0644 testuser testgrp
```

what the output will contain. An additional `pkgmap` file will be created. This is similar to prototype file and has additional information, such as checksum:

```
$ cd destdir
$ ls -R ZDJtest
ZDJtest:
install  pkginfo  pkgmap  reloc

ZDJtest/install:
checkinstall depend postremove copyright
➔ postinstall request

ZDJtest/reloc:
bin  etc  lib
ZDJtest/reloc/bin:
zdctest
ZDJtest/reloc/etc:
zdctest.dat  zdctest.html
ZDJtest/reloc/lib:
libzdctest.so
```

Adding the package

On the target machine you can run `pkgadd -d pkgdir` to add the package. You must have root access for adding the package. Use `pkgrm` for removing it. You can use the `pkginfo` command to list all the information about the added packages.

Conclusion

Here we have given you a first-hand introduction to package creation in Solaris. Once you're familiar with the steps, try the advanced options, such as classes and installing drivers. 

Solaris moves towards open source

by Clayton E. Crooks II

Sun Microsystems has decided to make publicly available the source code of the Solaris operating system under what's being called a community-source license. This move, which has been discussed openly for

the last year, is apparently an attempt to curb some of the attention being directed at Linux.

The term *community-source* simply means that Sun makes the source code for a product publicly available so that developers can

download the code free of charge and make changes to it, as long as they report back to Sun about any bugs they encounter.

The community-source concept falls short of being a true open-source project. The Linux community first popularized this type of arrangement when anyone could access the software and alter it, whether for their personnel development or commercial use. Under the Sun community-sourcing model, developers will continue to pay license fees to Sun if they decide to use any Solaris code in commercial products. If this were a true open-source project, Solaris would relinquish all intellectual property rights, meaning that Sun couldn't derive any financial rewards for having created the OS.

Sun's goal is to mimic the success of the completely open source Linux operating system, which benefits from enhancements suggested by volunteer programmers around the world. Sun will also allow programmers to download the source code. Linux is free for commercial as well as private use, but developers must make public all changes they make to the source code.

Because of the complicated licensing scheme, it appears that most users are unlikely to see any immediate changes in the way they acquire or pay for Solaris products. In addition, the developers really have very little to gain by altering the Solaris system as they may have difficulties in getting their code copyrighted and protected under GNU General Public Li-

cense covering open source code. One thing is certain: developers will gain knowledge by browsing through the source code.

This type of license is described as a combination of an industry standard proprietary license (typically an execution-only-license) and open-source licensing, which allows execution and access to source code with the right to improve and extend the source code. You can find more details at the Sun community source Web site located at www.sun.com/communitysource.

It's also possible that in the near future Sun may go all the way and make Solaris available as open source software. According to the *Wall Street Journal*, which quoted Greg Papadopoulos, Sun's chief technology officer saying that Sun only sees an upside in making all of the Solaris code available.

The first beta versions of the forthcoming Solaris 8 began shipping in September 1999, and Sun said the final product will be available early this year.

In order to add value to their office productivity suite, StarOffice will also follow the community-sourcing route so that developers can give their input and feedback on the software to Sun as a basis for future versions of the suite.

At this time, it doesn't appear that the community-source model will alter how most of us work and do business with the Solaris OS. We can only wait and speculate on how or when this movement will take off. 

Code coverage and profiling with Tcov

by Hariharan S

Nowadays, code coverage analysis has become an integral part of software development. Using code coverage tools, we can find out which part of our program is getting executed more often and then optimize those frequently executed parts for

better performance. In this article, we'll take a look at one such tool—Tcov.

Tcov, the code coverage tool

Using Tcov, we can produce a detailed output listing the percentage of code executed, number

of blocks executed, number of times a particular line is executed, etc. You can also execute Tcov multiple times.

There are two implementations of Tcov available, namely the old style and the new style. Let's look at both of them. To do so, we'll use the sample program coverage.c shown in **Listing A**.

Old-style implementation

For the old-style implementation, the procedure is to first assign a directory as TCOVDIR:

```
$export TCOVDIR="$HOME/tcovdir"
```

Next, compile the file with -xa option:

```
$ cc -xa coverage.c -o cover
```

Finally, run the program:

```
$ cover
```

This produces a file called coverage.d in \$TCOVDIR. Tcov uses the coverage.d file for creating the final output. Don't edit this file manually. Our sample code produced the following results:

```
$ cat $TCOVDIR/coverage.d
7 1
10 9
11 5
13 4
15 1
16 0
18 1
```

The next step is to run Tcov:

```
$ tcov coverage.c
```

This creates the final output file coverage.tcov, as shown in **Listing B**.

Listing A: Coverage.c—sample file used for demonstration

```
#include <stdio.h>

int main() {
    int i;
    for(i=1;i<10;i++)
    {
        if(i % 2 )
            printf("Odd Number\n");
        else
            printf("Even Number\n");
    }
    if( i < 10 )
        exit(-1);
    else
        exit(0);
}
```

Listing B: The Tcov output for our coverage program

```
#include <stdio.h>

int main() {
    1 -> for(i=1;i<10;i++)
    {
        9 ->   if(i % 2 )
        5 ->   printf("Odd Number\n");
        else
        4 ->   printf("Even Number\n");
    }
    1 ->   if( i < 10 )
    ##### ->   exit(-1);
    else
    1 ->   exit(0);
}
```

```
Top 10 Blocks
Line Count
10 9
11 5
13 4
7 1
15 1
18 1

7 Basic blocks in this file
6 Basic blocks executed
85.71 Percent of the file executed

21 Total basic block executions
Average executions per basic block
```

New-style implementation

The new-style implementation has a similar set of steps. First, compile the file with xprofile option:

```
$ cc -xprofile=tcov coverage.c -o cover
```

Next, run the program:

```
$ cover
```

This creates a directory called cover.profile and a Tcovd file in that directory. If TCOVDIR is set, this directory will be created under TCOVDIR. Tcovd is similar to the coverage.d file. There's no need to set TCOVDIR for this approach:

```
$ cat cover.profile/tcovd
TCOV-DATA-FILE-VERSION: 2.0
OBJFILE: /home/test/cover
TIMESTAMP: 942397214 957651
SRCFILE: /home/test/coverage.c
7 1
10 9
11 5
13 4
15 1
16 0
18 1
```

Note that the Tcovd file has some additional details. Next we have to run Tcov:

```
$ tcov -x cover.profile coverage.c
```

This creates the output file `coverage.c.tcov` in the current directory (or in `TCOVDIR`, if it's set). This will be the same as the `coverage.tcov` file. Unfortunately, the new style approach may not work in earlier versions of Solaris.

Output analysis

Now let's examine the output of Tcov in Listing B. Each executable line in the code has an arrow mark and a number or #### mark before it. The number indicates the total times that particular line is executed. #### means that the line isn't executed. The output will also have the top 10 blocks of the file and percentage of the code executed. You can run Tcov any number of times and

the database will update every time you execute the program.

Limitations

Even though Tcov is a useful tool, it does have limitations. You can't use the old-style implementation of Tcov for multithreaded/multiprocessing applications. You can use the `-xprofile=tcov` option, but the output count may be wrong (that is, it will say whether a line is getting executed or not, but the number of times it gives may be wrong). The program must exit properly for the `tcov.d` file to be saved. Also, Tcov doesn't support files with `#line` and `#file` directives.

Conclusion

Tcov will prove to be a very useful tool for bigger projects where manual analysis may be impossible or very difficult. Optimizing your program is more important than simply testing it. 

Making cron jobs quiet

I have a cron job that runs every five minutes to check whether an important daemon is running on my server. The script referred to in the cron job also restarts the daemon if it isn't running. Unfortunately, due to a recent upgrade, the daemon dies frequently, and every time the daemon is restarted by the cron job, I receive an email message telling me that it has been restarted. This is very annoying. How can I turn off this feature?

By default, cron jobs that produce output will redirect it in the form of an email to the user who owns the job. This output often looks like:

```
From pwatters Thu Sep 2 18:05:00 1999
Date: Thu, 2 Sep 1999 18:05:00 -0500
From: pwatters (Paul Watters)
Message-Id: <199909027656.ABC76584@ultra2>
To: pwatters
Subject: Output from "cron" command
Content-Length: 115
```

```
Your "cron" job on ultra2
/usr/local/scripts/checkdaemon
```

produced the following output:
checkdaemon: daemon restarted

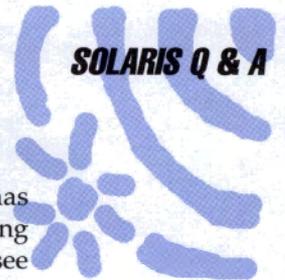
In this example, the `checkdaemon` script has restarted the daemon process after discovering that it wasn't running. If you don't want to see this output at all, then it's possible to redirect it to the null file `/dev/null`, which discards any data written to it. A cron entry in this case would look like:

```
5,10,15,20,25,30,35,40,45,50,55 * * * 1-5
/usr/local/scripts/checkdaemon > /dev/null
```

If the output was important for monitoring purposes, we could also redirect the output to a specific file for later review by a system administrator:

```
5,10,15,20,25,30,35,40,45,50,55 * * * 1-5
/usr/local/scripts/checkdaemon >>
/var/log/cron_errors
```

Paul A. Watters
Contributing Editor



It's not a bug, it's a feature

I'm new to Solaris, and I've just installed Solaris 7 on my Ultra-5 at the office. However, I tried to log on from home using telnet, and although I'm sure I used the right password for root, I get the message "Not on system console." I'm then disconnected. Help!

Since Solaris has a strong emphasis on security, many applications are shipped with default security measures enabled. Directly connecting as the root user to your machine using telnet means that you're transmitting your username and password in clear text. This may be visible to root users of other machines that make up the route between your

home computer and your office computer, allowing them to break into your computer as a privileged user. This is why telnet logins for root are turned off by default. You should consider using Secure Shell (www.datafellows.com) instead.

However, if you really want to take the risk, all you need to do is comment out the following line in the file `/etc/default/login`:

```
CONSOLE=/dev/console
```

Paul A. Watters
Contributing Editor

Network management for free

I'm designing a large network of Sun servers and other devices, but my budget is fairly tight for buying expensive networking management software. I need to be able to monitor the status of applications and hardware on remote machines. Can you suggest any software packages that are easy to use and install?

Most network management packages are based around SNMP, the Simple Network Management Protocol. Contrary to the protocol name, SNMP-based products are usually quite difficult to install and configure, unless there's an automated process for service discovery and maintenance.

One software product for Solaris is Enterprise SyMon, which is a Java-based server and console that can remotely manage Sun hardware components and applications running under the Solaris operating system. SyMon provides an integrated GUI to manage devices

that are SNMP-aware. The best news is that SyMon is available for free from

www.sun.com/symon/download/index.html

There's even a Windows 95/98/NT client if required.

If you're more interested in a freeware implementation, because you want to review the source code to see how it's all done, then the UCD implementation is available at:

ucd-snmp.ucdavis.edu

There are also graphical tools written in tcl/tk that can improve upon the standard text interface for this product.

Paul A. Watters
Contributing Editor

Solaris Device Configuration Assistant

I recently had a hard disk failure on my Solaris 7 x86 system, but when I tried to use the boot floppy disk, it no longer worked. Do I have to order a new one?

Fortunately, the Solaris Device Configuration Assistant (`boot.zip`) is available for download at soldc.sun.com/support/drivers/boot.html, as part of the Solaris Developer Connection.

There are also driver updates available at the same location. You can then copy the new boot file to a floppy disk using `dd` (on another Solaris machine), or by downloading the Windows version of `dd`. After rebooting the machine, insert the floppy disk, and you should be able to reconfigure your system.

Paul A. Watters
Contributing Editor

Why is this machine so slow?

by Mayank Arya

Of all the reasons that make your Solaris machine run slowly, CPU loading is one major contributor. Here's how I monitor my machine CPU loading:

```
/usr/ucb/ps -aux (not the /bin/ps)
```

This lists out all the processes currently running on my machine, along with useful information, such as how many CPU cycles are being used by each process. This list is sorted already in terms of CPU usage, with the process using most CPU appearing first.

"But there's a little problem here," you might say (assuming you have a whole lot of processes running on the machine). "The list is too long." The good news is that the information we're interested in is right at the top, so using the head command helps:

```
/usr/ucb/ps -aux | head -10
```

You might like to put the whole command as an alias, something like `cpu-load`, in your `.cshrc` file (in your home directory) to save you from typing this long command time and again. Adding this line to your `.cshrc` will do it:

```
alias cpuload "/usr/ucb/ps -aux | head -10"
```

So the next time you think something is chewing up your CPU, type the same `cpuload` and deal with the process loading the CPU.

About our contributors

Clayton E. Crooks II is a self-employed computer consultant living in Knoxville, TN. He is married with one child. His hobbies include game development, 3-D modeling, and any athletic activity he can find time for.

Edgar Danielyan is currently self-employed. His list of qualifications include Cisco Certified Network Associate, Diploma in Company Law from the British Institute of Legal Executives, and certified paralegal from the University of Southern Colorado. He has been working as a network administrator and manager of a top-level domain of Armenia. He's also worked for the United Nations, the ministry of defense, a national telco, a bank, and has been a partner in a law firm. He speaks four languages, likes good tea, and is a member of ACM, IEEE CS, USENIX, CIPS, ISOC, IPG, etc. He can be reached at edd@danielyan.com.

Don Kuenz works at Computing Resources Company (gtcs.com/crc). Computing Resources Company provides programming, administration, and hardware for Sun and IBM platforms. You can reach Don at kuenz@gtcs.com.

Hariharan S works as a software engineer at Wipro Infotech Ltd. in Bangalore, India. He can be reached at hharan@mailcity.com.

Paul A. Watters is the project manager at Neuroflex, where he's responsible for developing natural language database systems in Java on the Solaris platform. He can be reached at pwatters@mpce.mq.edu.au.

Inside Solaris (ISSN 1081-3314) is published monthly by
ZD Journals 500 Canal View Boulevard, Rochester, NY 14624.

Customer Relations

US toll free(800) 223-8720
Outside of the US(716) 240-7301
Customer Relations fax(716) 214-2386

For subscriptions, fulfillment questions, and requests for group subscriptions, address your letters to

ZD Journals Customer Relations
500 Canal View Boulevard
Rochester, NY 14623

Or contact Customer Relations via Internet email at journals@mail.training.com.

Editorial

EditorGarrett Suhm

Assistant EditorJill Suhm
Managing EditorMichelle Rogers
Copy EditorsRachel Krayer
Christy Flanders

Contributing EditorsClayton E. Crooks II
Edgar Danielyan
Don Kuenz
Hariharan S
Paul A. Watters

Print DesignerMelissa Ribaud

You may address tips, special requests, and other correspondence to

The Editor, *Inside Solaris*
500 Canal View Boulevard
Rochester, NY 14623

Editorial Department fax(716) 214-2387

Or contact us via Internet email at inside_solaris@zdjournals.com.

Sorry, but due to the volume of mail we receive, we can't always promise a reply, although we do read every letter.

ZD Journals

General Manager Kelly Baptiste
Manager of Customer Relations Heather Loicono
Manager of Operations Cristal Haygood
Manager of Print Design Charles V. Buechel
Manager of Product Marketing Mike Mayfield
Senior Product Marketing Manager Brian Cardona

Postmaster

Periodicals postage paid in Rochester, NY and additional mailing offices.

Postmaster: Send address changes to

Inside Solaris
P.O. Box 92880
Rochester, NY 14692

Copyright

Copyright © 2000, ZD Inc. ZD Journals and the ZD Journals logo are trademarks of ZD Inc. *Inside Solaris* is an independently produced publication of ZD Journals. All rights reserved. Reproduction in whole or in part in any form or medium without express written permission of ZD Inc. is prohibited. ZD Journals reserves the right, with respect to submissions, to revise, republish, and authorize its readers to use the tips submitted for personal and commercial use. For reprint information, please contact Copyright Clearing Center, (978) 750-8400.

Inside Solaris is a trademark of ZD Inc. Sun, Sun Microsystems, the Sun logo, SunSoft, the SunSoft logo, Solaris, SunOS, SunInstall, OpenBoot, OpenWindows, DeskSet, ONC, and NFS are trademarks or registered trademarks of Sun Microsystems, Inc. Other brand and product names are trademarks or registered trademarks of their respective companies.

Printed in the USA.

Price

Domestic\$99/yr (\$9.00 each)
Outside US\$119/yr (\$11.00 each)

Our Canadian GST# is: R140496720. CPM# is: 1446703.
QST# is: 1018491237.

Back Issues

To order a back issue from the last six months, call Customer Relations at (800) 223-8720. Back issues cost \$9.00 each, \$11.00 outside the US. You can pay with MasterCard, VISA, Discover, or American Express. Collections of back issues are available on CD-ROM as well. Please call for more information.

Are you moving?

If you've moved recently or you're planning to move, you can guarantee uninterrupted service on your subscription by calling us at (800) 223-8720 and giving us your new address. Or you can fax us your label with the appropriate changes at (716) 214-2386. Our Customer Relations department is also available via email at journals@mail.training.com.

- Using CDE
- Packet filter firewalls
- Free Enterprise Javabeen servers

Please include account number from label with any correspondence.

Dual booting Solaris and Linux

by Clayton E. Crooks II

Have you ever wanted to have Solaris and Linux installed on the same machine? The following steps will guide you through installing the Solaris operating environment in partition 0 and Linux in partition 8, allowing you to boot to the OS of your choice.

Installing Linux

The first step is to install any version of Linux. This can be from a CD-ROM or the Internet.

First, when the installer prompts for partitioning, allocate partition 8 for Linux root and partition 7 for Linux swap. Allocate partition 1 for Solaris (SunOS root) and partition 2 for SunOS swap. You can also allocate other partitions as desired for Solaris or Linux use. The Linux installer calls the first partition 1, and has codes for the Solaris partitions. Partition 3 (or 4) should be the whole disk.

Second, make partition 8 the root partition, and install Linux there. Install Linux swap in partition 7. Third, complete the Linux install as usual. It's important that you install the silo boot loader in the same partition as the Linux root. Let it put the Silo location in `nvalias`; you can change it later. It should show up as boot-device `disk:h` in `printenv` at the OK prompt on the Ultra systems.

Fourth, boot the system with Linux to check install (startx will get the X windows up). Finally, halt will sync the system and halt the OS (OK prompt).

Installing Solaris

Now you can install Solaris from the Internet or CD-ROM. First, during the Solaris partitioning, the installer will ask if you want to preserve data. Click Preserve Data, and then preserve all the partitions used for Linux.

Second, create one or more partitions for use with Solaris, and let Solaris format them. This is optional. Next, install Solaris on partition 0 (first partition). When the installer asks if you want to make the new root partition the default boot in NVRAM, answer yes.

Finally, the installer will then complete the Solaris install as usual, and will reboot automatically to Solaris if you ask it to do so.

Setting up your boot alias

You now can set up aliases to allow you to dual boot. At the OK prompt, type `show-disks`. The disk's paths will be printed. Type `devalias` to get the path format for the disk you are using. Now follow these steps for setting up each boot option:

- Choose the correct hard disk path and type `nvalias linux ^Y <disk_path_from_above>@0,0:h`.
- If you wish, type `nvalias solaris ^Y <disk_path_from_above>@0,0:a`.
- To fix the `printenv`, type `setenv boot-device disk:a`. To change the default to Linux, substitute `:h` for `:a`.
- Set autoboot on if you wish.
- Use `boot`, `boot solaris`, or `boot disk` to boot Solaris from partition a. `boot linux` boots Linux from partition h. You can also reinstall Linux or Solaris as many times as you want without altering the other installation.

That's all there is to it. If you're in need of additional or more advanced options, you might want to consider purchasing a boot manager such as Bootit from Terabyte Unlimited at www.terabyteunlimited.com.