Volume 6, Number 9 • September 2000
www.elementkjournals.com/sun
US $11.00

Inside Solaris™
Tips & Techniques for users of Sun Solaris

Porting applications between NT and UNIX

by Clayton E. Crooks II

With so many companies using a mixed computing environment, it's becoming increasingly difficult to develop software for a single platform. As a result, many developers are choosing to develop applications for multiple environments. This may take the Operating System (OS) decision out of the equation, but developing applications for multiple environments can be a very costly and time-consuming process.

A number of proprietary GUI and application builder products support both NT and UNIX, and you can use these exclusively in a project if your goal is to develop an entirely new application. However, if you have an application that already runs in one environment, you probably don't want to start over completely, so you're left with a decision to port your application to the other OS.

NT to UNIX

A seemingly unending list of problems occurs when porting applications from one environment to the other, and the process of going from NT to UNIX is not immune to these difficulties. However, a software application called MainWin XDE (XDE is an acronym for eXtended Development Environment) from Mainsoft Corporation (**www.mainsoft.com**) makes this difficult process manageable, and is almost a necessity for porting NT applications to UNIX. Their approach to this challenge is shown in **Figure A**.

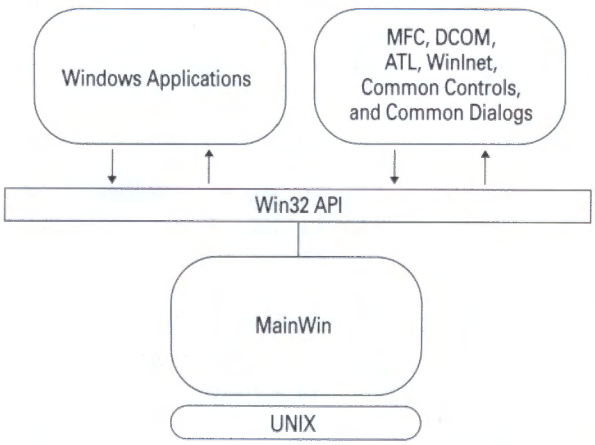


Figure A: *Mainsoft uses an implementation of the win32 API on UNIX.*

The tool has undergone several changes, and the latest release is outstanding. With help from several million lines of Windows NT source code, which was obtained from Microsoft, several new features have pushed this product to the head of the class. The Mainsoft and Microsoft relationship is key to this software, and both sides have benefited tremendously from their partnership. Mainsoft obviously benefits from the Microsoft source code and knowledge, but the

In this issue

Quick Tips:

- Send output and errors where you want
- Use type to find commands and learn more about them
- Why making copies of UNIX CDs doesn't work

elementk JOURNALS
A Division of Element K Press

relationship is far from one-sided. In fact, Microsoft used MainWin to port Internet Explorer 4.0 to UNIX.

MainWin functionality

With the support of the new source code, developers can now incorporate Microsoft functionality, such as Distributed COM and National Language Support. MainWin also supports the latest release of MFC and the Windows NT Shell. Controls such as Internet Explorer 4.0 common, common dialogs, and an NT edit control are also available.

MainWin features a full set of the Windows API, including support for structure exception handling, asynchronous procedure call APIs, Windows NT 4.0 memory management, and shell 32 libraries. Mainsoft has also ported Microsoft's window manager to the UNIX world, which results in a genuine Windows NT look and feel.

COM and ActiveX support

One of the biggest problems when porting NT applications to UNIX is the lack of support for COM objects. The newest release of MainWin has alleviated this problem. It now includes the ActiveX Template Library, which is designed to allow developers to create compact and reusable COM objects. Previously, a developer would have had to rewrite every COM or ActiveX-based component for use in UNIX environments. This was the single biggest headache for developers porting NT applications to UNIX, and it has been eliminated completely by MainWin.

With the release of a newly revamped version of its NT-to-UNIX porting tool, Mainsoft Corporation hopes to offer an entire solution for IS managers charged with bringing Windows NT applications to UNIX platforms such as Sun Solaris, SGI IRIX, and HP-UX. While many outstanding features exist, the single biggest advantage is that developers can leverage Windows NT development standards like ActiveX on any supported UNIX platform.

UNIX to NT

The NuTCRACKER package is an excellent solution for the reverse objective, porting applications from UNIX to NT. Mortice Kern Systems Inc. (www.mks.com) purchased NuTCRACKER from DataFocus, and it appears that the move has taken NuTCRACKER to the next level. MKS already had an excellent product called MKS Toolkit, and combining the development package with it provides a complete solution for UNIX-to-NT porting.

As we previously mentioned, Microsoft played a role in the development of MainWin, so it should come as no surprise that they would be involved with the reverse application. The agreement, originally between DataFocus and Microsoft, created a set of Visual C++ wizards that developers could use to transform existing UNIX-based code into 32-bit COM components. Microsoft's help has been acknowledged, but the technology they provided to DataFocus has never been specified.

NuTCRACKER features

NuTCRACKER provides four different solutions for UNIX/NT interoperability. The first two of these solutions don't play a big role in the porting of applications, which is what we're most interested in. As a result, we'll only mention them briefly so that we can move on to the last two.

The first solution is a set of UNIX shells and hundreds of utilities that work under NT. For instance, instead of using `dir` at a command line, you can use `ls`. The complete shells allow you to perform shell scripting, including using any of the provided utilities. The second is an interoperability tool, which allows you to easily move back and forth between UNIX and NT environments.

Our primary focus for this article is on application development, which is the objective of the last two solutions provided by NuTCRACKER. The third solution provides 2,700 APIs for porting to NT's development platform from a number of UNIX versions. The final aspect of NuTCRACKER is a set of conversion utilities that let you update legacy UNIX applications to take advantage of current OS features. NuTCRACKER is language independent, so you can port code written in C, C++, FORTRAN, Ada, and COBOL.

The Operating Environment (OE) that provides the UNIX subsystem under Windows is the core of NuTCRACKER. It has been constructed out of a number of DLLs (Dynamically Linked Libraries) and a service manager that handles the processes. The OE has several options you can choose, including X servers from WRQ and SCO, XRT's PDS, Wintif (provides Motif look and feel), Hummingbird's OpenGL, and a Telnet server.

Near perfection

NuTCRACKER is one of the few utilities of its type (or of any type, for that matter) that does everything it claims. While the software does have a learning curve that can take weeks to master, it's relatively painless to work with. NuTCRACKER is well documented, and should be used by anyone looking for a UNIX-to-NT solution.

Final thoughts

With the increase in mixed computing environments, porting applications to NT or UNIX is

something that you'll definitely see more of in the future. Depending on your needs, MainWin and NuTCRACKER take most of the burden off the porting process. Each of the packages has involved learning curves, but a few weeks of use will make even a beginner comfortable with the solutions they provide. In fact, once you have struggled successfully through the process a few times, you'll find that porting simple applications can take as little as a couple of hours. Obviously, large applications will take a longer period of time and will vary according to many different variables. If you're ever confronted with the process of porting applications, we believe you'll find these solutions to be up to the challenge for even the most difficult of situations. ✳

Squid basics

by Alan Orndorff

When the World Wide Web was first created, it used a simple mechanism to display Web pages. You pointed your browser at a Web site, and your browser displayed the page on your screen. In the early days, this was more than sufficient. As time rolled on, however, the number of Web pages, and the number of people viewing those Web pages, exploded. 56K modems weren't invented yet, and access speeds were not what they are today. The constant fetching and rendering of Web pages takes bandwidth and many people using the same corporate network to access the Web slows down access times for everyone. Then came the proxy server.

There are a variety of proxy servers available. In this article, however, we're going to look at Squid, which has proven to be very popular. It also has the added bonus of being free. Before we look at Squid, we'll give you a brief overview of what a proxy server is.

Proxy server overview

Instead of your browser accessing the Web directly, your Web browser now points to a proxy server. It's then the job of the proxy server to fetch the Web page for you and send the results back to your browser. The proxy server also has its own caching mechanism. When someone retrieves a page from the Web, it's stored in the proxy server's cache. When another person wants to access the same Web site, the proxy server does not go to the Web again to retrieve the same page, but instead sends the page from its cache. This conserves precious Internet bandwidth and can save substantial fees for large network pipes.

Proxy servers may sound like a great idea for large corporations, but a corporation of any size will benefit from using one. Some proxy servers can also pre-load Web sites into their cache without anyone actually going to the Web site beforehand. This is a great time saver, as your proxy server can cache a Web site at night, and then the following day you can access it at your LAN's speed without having to connect to the Internet.

Your Web browser has its own mechanism for storing retrieved Web pages. The browser simply keeps recently used Web files on your local hard drive. But by using a proxy server, it's as if everyone is using the same cache.

Another function of a proxy server is access control. You can easily restrict who can and can't access the Web, as well as limiting access to certain sites, or logging every site that a user accesses.

Despite this capability, everything on the Web isn't cacheable. Forms or database lookups are generally not cacheable. A Web page designer can also mark any Web page as non-cacheable as well. Further, all documents in the proxy's cache have expiration dates. Many of the Web pages on the Internet are not static and are in a constant state of change. If the proxy server doesn't expire the page, you may never know that the Web page has changed. Generally speaking, a proxy server is usually configured with two network interfaces. One is used to service the internal network and the other will point to a firewall or the Internet.

Getting Squid

You'll find Squid at **www.squid-cache.org**. If you're currently running Solaris 8 with a full installation, compiling Squid is straightforward. If you're running an older version of Solaris, you'll need to install PERL before compiling. You'll find PERL at **www.perl.com**.

After you've downloaded Squid, unpack the source by using `gzip` and `tar`. Then, type `./configure` to start the process of compiling Squid. You

may want to use `./configure -help` to see the available options, such as how to set up SNMP support. Go through the list of options and select the ones you want before compiling the proxy server.

After the configuration finishes, simply type `make` and then `make install` and you're ready to start configuring your new proxy server. All files end up in the /usr/local/squid directory hierarchy.

Server startup

In order to run Squid, you need to create the cache directories. You do this by running the following command:

```
/usr/local/squid/bin/squid -z
```

Once you've created the cache directories, you're ready to start the Squid service. You can start the service and watch its output on your term window by running:

```
/usr/local/squid/bin/squid -d1
```

If you've properly configured Squid, you'll see the line "Ready to Service Request." To start Squid as a daemon, run it with no options.

When Squid starts, it spawns two processes. The first is the parent process, and the second is the child process. The job of the parent process is to ensure that the child process is running. If the child process exits, the parent process will restart it.

To stop Squid, run the following:

```
squid -K shutdown
```

The configuration file for Squid is squid.conf, and the file is heavily commented, so configuring Squid is relatively easy.

Basic parameters

There are some basic things you may want to change in squid.conf immediately. These are defined in Table A.

> **Note:** Do not run Web servers, proxy servers, or other services that are accessible to the Internet as root if you can help it.

Set up these services with the least user privilege that will allow them to run. Then if someone does compromise the server, the damage will be minimal.

Log files

When things are up and running and moving along just fine, everyone is happy. But all things break, and we're here to fix them. Log files are an important part of troubleshooting problems. In Squid, some log files have to be included in the `./configure` command, or they won't be available to you. Some important log files are:

- **cache_log.** This log file contains the debug and error messages that Squid generates. If you start Squid with the `-s` option, a copy of certain messages will be sent to syslog.
- **access_log.** If you want to know which sites your end users are visiting, then take a look at this file. As previously mentioned, this can be in Squid native format or stored similar to a Web server's log file. The native file format is preferred for log analysis programs.
- **swap.state.** This file is a record of every cache object written to the disk. It's read on server startup to reload the cache. Removing this file while Squid is running effectively tells Squid that no objects are currently cached. If you remove swap.state while Squid is running, you can create a new one by running:

  ```
  squid -K rotate
  ```

You can also tell Squid to shut down and it will rewrite this file before it exits.

If you remove this file while Squid is stopped, it will scan all of your cache_dir directories and re-create the file. This can prolong the time needed to start Squid.

Rotation

If your log files are getting too big, rotate them before archiving them. To rotate your log files, set up a cron job or manually run:

```
squid -K rotate
```

You can also configure squid.conf to rotate the log files for you.

Cache manager

The Cache Manager is a cgi utility for displaying statistics about the Squid process as it runs. Its setup is covered in greater detail in the Squid FAQ. By using the Cache Manager, you can view the statistics of your Proxy Server through a Web page.

Access control

You can configure access control with Squid in several ways. The first option is to set up user accounts and have users enter a username and password before being allowed to use the proxy server for outbound connections. Second, you can allow

Table A: *Use these parameters in quid.conf to configure Squid*

Parameter	Descriptions
`http_port`	Defaults to 3128. Other proxy servers default to 8080, but its assignment is up to you. If you have a multi-homed machine, you can configure Squid to listen on all interfaces or only on certain interfaces.
`tcp_outgoing_address`	Sets the outgoing IP address that the proxy server will use to communicate to the outside world.
`maximum_object_size`	Defaults to 4 MB and specifies the largest object size that the server will cache. Remember, a proxy server caches more than just Web pages. It can also cache ftp requests. Anything bigger than the value set here won't be cached.
`cache_dir`	Specifies where on the disk the cache files will be stored. This specifies the top-level directory for Squid's cache. You can specify multiple `cache_dir` directives to split the cache among multiple slices or mount points. This directory must pre-exist, as Squid will *not* create it. Make sure that the user account that you'll run as Squid has access to these directories. If no `cache_dir` is specified, Squid defaults to /usr/local/squid/cache. You also set the size of the cache along with the directory location.
`cache_access_log`	Contains an entry for every http request.
`cache_log`	Specifies where general information about the proxy server is stored.
`emulate_httpd_log`	Makes the log format similar to a Web server when you set this option to on.
`pid_filename`	Sets the filename where the process ID is written to; may be set to none.
`ftp_user`	Allows you to set the anonymous user ID that Squid will use when accessing an anonymous ftp server.
`ftp_list_width`	Allows you to set the width of ftp listings that will appear on the user's screen. Setting this value too low can make it hard to read directory listings.
`quick_abort_min`, `quick_abort_max`, and `quick_abort_pct`	Sets directives that help with handling impatient end users. When the end user aborts a request, Squid checks these values to determine what to do next. If the transfer has less than the `quick_abort_min` remaining (in KB), then the transfer will finish. If it has more than the `quick_abort_max` remaining, then Squid abandons the retrieval of the Web page. If more than `quick_abort_pct` (percentage) has completed, Squid finishs the transfer. Setting the value of `quick_abort_min` to -1 can disable this function.
`connect_timeout`	Indicates how much time Squid will wait for a connection to complete. If the transfer of the Web page doesn't occur within this time frame, the connection aborts.
`pconn_timeout`	Indicates the maximum time that a persistent idle connection will be allowed.
`cache_mgr`	Indicates the email address that administrative alerts will be mailed to.
`cache_effective_user` and `cache_effective_group`	Allows you to set the username and group that you want the Squid process to run as. If you start the process as root, it immediately switches to the values specified here. Otherwise, it defaults to the username/group that you are currently running as.

anonymous access through the proxy server and merely block traffic to certain sites. Your third option is to simply set up the proxy server to allow anyone to go anywhere. Finally, you can use a combination of the above.

Setting up Squid to allow for usernames and passwords is fairly straightforward. You simply compile another program to take the username and password entered by the end user, and the program returns either an "OK" or an "Err" to indicate access granted or access denied. To make this process easier, a program is included with Squid to do this for you. You can also have an LDAP, SMB, mysql, PAM, or Radius authenticator program validate usernames instead of the supplied program. All of this is discussed in detail at:

home.iae.nl/users/devet/squid/proxy_auth/

Don't worry that the Web page is not very long. You basically tell Squid which authentication program to use, where it stores its passwords, and how many child processes will respond to username and password queries.

The next thing you need to be aware of is the acl tags. The basic format of an acl tag is as follows:

```
acl aclname acltype string or
acl aclname acltype file
```

When using `file`, it should specify one item per line. The term `acl`, in the above code, is generic and merely indicates what follows is the `acl` rule. `aclname` is up to you and is used with the access/deny `acl` rules. `acltype` is one of the following:

```
src/dst
ip-address/netmask or addr1-addr2/netmask
```

You can also specify `myip` instead of `src/dst`:

```
srcdomain/dstdomain
.foo.com

srcdom_regex/dstdom_regex
 [-i] regular expression
```

Listing A: *The regular expressions used by Squid*

```
time:
[day-abbriv] [h1:m1-h2:m2]

S-Sunday
M-Monday
T-Tuesday
W-Wednesday
H-Thursday
F-Friday
Saturday

h1:m1 < h2:m2

url_regex
[-i] expression

urlpath_regex
[-i] expression

port
80 443 etc...

myport
   localhost port

proto
   HTTP FTP

method
   GET POST

browser
   [-i] Regular Expression
```

Note that regular expressions are case sensitive, unless you specify otherwise by including the `-i`, as shown in **Listing A**.

Once you've defined your access control lists, you need to do something with them. The next items that you'll want to set up are the `http_access` rules. These are easy to set up and use the syntax of the following:

```
http_access allow|deny [!]aclname
```

For example, if you want to try to prevent your employees from shopping online, you could define the following access control rules:

```
acl all src 0.0.0.0/0.0.0.0
acl SSL_ports port 443

http_access deny SSL_ports
http_access allow all
```

This would create the acls `SSL_ports` and `all`. Any attempt to access port 443 would be denied, but access to all other Web sites would be allowed.

You should keep in mind two important points while assigning acls and access entries:

- All elements of an acl entry are ORed together.
- All elements of an access entry are ANDed together.

If you had the following defined, it would result in no access:

```
acl ten src 10.1.1.0
acl eleven src 11.1.1.0

http_access allow ten eleven
```

In order for someone to go through the proxy server, he'd need to have an address on the 10.1.1.0 network *and* the 11.1.1.0 network. In order to allow both networks to go through the proxy server, you'd rewrite the `http_access` rules as:

```
http_access allow ten
http_access allow eleven
```

You'll find more examples on setting up acls in the Squid FAQ or by reading the squid.conf file.

httpd accelerator mode

You can also use proxy servers to speed up access to your internal Web site. When someone points his browser at your Web site, say www.example.com,

the request would go to the proxy server instead of to your Web server. Then the Squid process would decide whether to send the page to the end user from its cache, or to forward the request on to the real Web server.

Using such a setup, you could place your proxy server outside of the corporate firewall and have it make calls to your internal network through firewalls. Another option would be to set up Squid as the first machine inside of your firewall and have it send requests to your Web servers. You should not set up a machine as both an httpd accelerator and a proxy server.

To make Squid your first machine internally, set your `http_port` in squid.conf to port 80, the same one your Web server would normally use. Then, set `httpd_accel_host` to your Web server's dns entry, and `httpd_accel_port` to its port number. If you're using virtual hosting, be sure to set `httpd_accel_host virtual` in your squid.conf file. If you really want to run in both httpd_accelerator mode and proxy server mode, then add `httpd_accel_with_proxy on`.

DNSSERVER

DNSSERVER is a process forked by Squid to resolve IP addresses from domain names. Because host lookups can block the main Squid process, these child processes are used to handle IP resolution. We recommended having enough DNSSERVER processes to handle your expected load, or Squid will stop occasionally.

The ftp client

The ftp client included with Solaris won't go out through a proxy server; however, some other clients can be configured to go through them. A couple of programs that you can configure to use proxy servers are GNU's wget and Gnome's Midnight Commander.

Blocking undesirable Web sites

Some corporations have very strict policies on Web access, while others are less restrictive. Blocking or not blocking sites is usually a corporate-wide decision. Be aware that even if you do block access to some sites, we've seen corporate employees with laptops request modem lines and connect to the Internet that way to avoid having their accesses logged by a proxy server. The Squid FAQ lists the following two sites to help you set blocks to less desirable Web sites:

www.hklc.com/squidblock

www.snerpa.is/notendur/infilter/infilteren.phtml

Conclusion

We've looked at the basics that you may want to consider while setting up Squid. To get a feel for all of the features of Squid, you should take a run through the Squid FAQ, as well as reading the squid.conf file. If you read through the FAQ, you'll find that paid support for Squid is available, as well as a few other features of this program. ✳

Sniff your own networks with tcpdump

by Boris Loza

A *sniffer* is any device, software, or hardware that listens to all packets traveling along a network. Bad guys use sniffers to breach your security by capturing and analyzing all network traffic. Good guys use sniffers to protect the network.

Today you can find many different types of this software. Some sniffers are used by crackers, and others by network administrators. Some sniffers are free, and some are expensive. Further, some are simple. There are companies who produce entire suites of sniffer applications designed to diagnose network problems. In this article, we'll show you how to use a freely available sniffer—tcpdump.

About tcpdump

The tcpdump program was written by Van Jacobson, Craig Leres, and Steven McCanne, all of the Lawrence Berkeley Laboratory at the University of California at Berkeley. Like any other sniffers, tcpdump captures packets on the network by placing a local interface in promiscuous mode. Normally, your network interface will ignore any pockets that aren't addressed to your system. By

placing your interface in promiscuous mode, tcpdump captures all packets, regardless of address, and allows you to examine their headers. The tcpdump program also allows you to extract particular types of network traffic based on header information.

Installing tcpdump

You can download tcpdump from **ftp.ee.lbl.gov/tcpdump.tar.Z**. The latest tcpdump version found here was 3.4. You can also download libpcap.tar.Z from the same destination (**ftp.ee.lbl.gov/**). This application is a system-independent interface for user-level packet capture, and will be used in conjunction with tcpdump. You must install libpcap first. After downloading files, type:

```
# zcat libpcap.tar.Z | tar xvf -
# zcat tcpdump.tar.Z | tar xvf -
```

Listing A: *Sample output from tcpdump*

```
#tcpdump
tcpdump: listening on hme0
11:17:36.965387  192.168.10.1.1028 > 192.168.10.5.700: udp 82
11:17:42.645580 pine.tree.com.34342 > birch.tree.com.telnet:
➥S 1819099388:1819099388(0)
win 8760 <mss 1460> (DF)
11:17:50.011072 pine.tree.com > oak.tree.com: icmp: echo request (DF)
11:17:50.011091 oak.tree.com > pine.tree.com: icmp: echo reply (DF)
11:17:55.870599 arp who-has 192.168.10.1 tell 192.168.10.55
^C
```

Table A: *Each record generated in our sample has the following fields*

Field	Data
Timestamp	11:17:36.965387
Source.Port > Dest.Port:	192.168.10.1.1028 > 192.168.10.5.700:
Protocol	udp
Bytes of data	82

Table B: *tcpdump generates the following record for the TCP protocol*

Timestamp	11:17:42.645580
Source.Port > Dest.Port	pine.tree.com.34342 > birch.tree.com.telnet:
Flags	S
Begin-Seq#:End-Seq#(Bytes)	1819099388:1819099388(0)
Options	win 8760 <mss 1460> (DF)

This extracts the tcpdump and libpcap distributions. Then, go to the libpcap-0.4 directory and read the INSTALL file. In most cases, it's enough just to type `./configure` followed by `make install`, `make install-incl`, and optionally `make install-man`. Doing so installs libpcap and the manual entry. Now you can go to the tcpdump-3.4 directory to start building tcpdump. Type `./configure` and build tcpdump by running `make`.

If everything builds okay, become root and type `make install` and `make install-man`. Doing so installs tcpdump and the manual pages. If you decide to install tcpdump in a directory other than /usr/local/sbin, edit the BINDEST path in Makefile.in and run `./configure`.

Tcpdump output format

Running tcpdump is easy. After you type `tcpdump` at root, you'll see the output shown in **Listing A**. You'll find that `tcpdump` generally produces one line of output for each packet that it sees. For each connection, `tcpdump` will always display (except in very special cases) a timestamp, a source IP address, a destination IP address, and some additional information about the packet (such as protocol and port information). Further, `tcpdump` has a default standard output based on the protocol. On the `tcpdump` output in **Listing A**, we can see the UDP, TCP, ICMP, and ARP protocol information.

Table A shows the descriptions of each field shown in **Listing A**. In this example, tcpdump is reporting traffic from host 192.168.10.1 on port 1028, to host 192.168.10.5 on port 700, using the UDP protocol. The direction of the traffic is indicated by the greater than symbol (>) between the source and destination address. The timestamp is in the format of `hour:minutes:seconds.millionth_of_seconds`. UDP packets may or may not have `udp` in the output.

Table B shows the fields that are present for a TCP packet. This is identical to the UDP record as far as timestamp, source and destination host, and port. What distinguishes the TCP format from the others are the TCP flags, sequence numbers, acknowledgements, acknowledgement numbers, and TCP options.

In this record, we see the flag `SYN` or `S` set following the destination port telnet. The `SYN` flag indicates the beginning of a telnet session. Other possible flags are `P` for `PUSH` that sends data, `R` – `RESET` that

Table C: *tcpdump generates the following record for the ICMP protocol.*

Timestamp	Source	>	Destination:	icmp:	icmp Message
11:17:50.011072	pine.tree.com	>	oak.tree.com:	icmp:	echo request (DF)
11:17:50.011091	oak.tree.com	>	pine.tree.com:	icmp:	echo reply (DF)

aborts a connection, and F – FIN that terminates a connection. If you see a period '.' in the flag field, it simply means that none of the PUSH, RESET, SYN, or FIN flags are set.

You'll see that 1819099388:1819099388 is the beginning:ending set of sequence numbers. The ending sequence number is the sum of the initial sequence number plus the number of TCP data bytes sent in this segment (in this case 0, which is why both numbers are equal).

Finally, there is a TCP options field. We see that pine.tree.com is advertising a window size of 8760 bytes, a maximum segment size of 1460 bytes, and no fragmentation required (DF – Do not Fragment option).

Table C shows a sample tcpdump ICMP output. *ICMP* is the protocol used for error control and message handling. There are many different types of ICMP records that have different messages. In the first ICMP record following the timestamp, we can see the source pine.tree.com. Following the greater than sign, we see the destination oak.tree.com. Because ICMP doesn't use ports to communicate like TCP and UDP, we won't see this information. The ICMP message type that follows the destination host in the first record is an ICMP echo request or ping. In the second record, ICMP message type is an ICMP message reply.

The last row of tcpdump output is the ARP request. This shows the IP address of 192.168.10.55 as the target IP address and the 192.168.10.1 IP address as the sender IP address.

There are also command-line options for tcpdump that will alter the default behavior, either by collecting specified records, printing in a more verbose mode, printing in hexadecimal, or writing records as raw packets to a file instead of printing as standard output. For all of tcpdump's options, consult the tcpdump(1) man pages.

Filters for tcpdump

As we mentioned earlier, tcpdump allows you to extract particular kinds of network traffic. The basic syntax for a tcpdump filter is as follows:

```
<header>[<offset>:<length>] <relation> <value>
```

So, to use this filter to detect telnet traffic, you can write the following:

```
#tcpdump tcp and dst port 23
```

The filter is `tcp and dst port 23`. Only those packets that satisfy the filter requirements will be captured. This filter selects those packets that have protocol type TCP and destination port number 23. Since the telnet protocol uses port 23, this filter selects telnet packets.

The filter syntax for `tcpdump` is very robust. You can employ the filters to extract connections involving specific networks, hosts, and ports. The simplest way to do this is to use the macros that tcpdump supplies for `<header>`, such as `src`, `dst`, `host`, `net`, and `port`. However, if you wish to specify a range of networks, hosts, or ports, you need to use relational operators in conjunction with the individual header byte specifications. As an example, you can specify all destination IP addresses belonging to the range 172.16.0.0 – 172.31.255.255 by writing the following:

```
dst net 172  and (ip[17] > 15) and (ip[17] < 32)
```

This specifies the first octet and the range for the second octet of the destination IP address. You could alternatively specify the first octet of the destination address by writing `ip[16] = 172`, but `dst net 172` seems a bit more intuitive. Similarly, you could specify the IP address range by writing:

```
(dst net 172.16 or dst net 172.17 or dst net
172.18 or ... or dst net 172.31)
```

However, this can quickly get cumbersome when a large number of networks are involved. Tcpdump accepts the regular relational operators for `<relation>`: `= < > <= >= !=` and the logical operators: `and`, `or`, and `not`. Further, `tcpdump` accepts either hostnames or IP addresses, and either port numbers or service names. Note that when you use service names, and tcpdump understands the protocol, both the port number and the protocol are checked, instead of just checking the port number.

Let's create several filters. To capture all traffic between pine.tree.com and oak.tree.com machines for a telnet port, you can write:

```
#tcpdump host pine.tree.com and host oak.tree.com
 and port telnet
```

Or just to detect telnet traffic with the following:

```
#tcpdump "tcp[2:2] = 23"
```

This works because the destination port number is two bytes, beginning at the second byte in the TCP header. Telnet uses `23/tcp` for the server port. We use single quotes to protect the square brackets from being interpreted by the UNIX shell.

Following are some more examples:

- **ICMP filter.** This filter looks for all ICMP packets that are not ping packets:

```
#tcpdump "icmp and icmp[0] !=8 and icmp[0] != 0"
```

- **r-utility filter.** `rlogin`, `rcp`, `rsh`, `rdist` and so forth. It is wise to look for packets with r-utilities from unknown sites:

```
#tcpdump "ip and (tcp dst port 512 or
   tcp dst port 513 or dst port 514)"
```

- **X11 filter.** This filter finds any X11 or Motif traffic:

```
#tcpdump "tcp and (port 6000 or
   port 6001 or port 6002)"
```

- **Inbound SYNfilter.** This filter monitors any SYN connections to ports you are not expecting traffic on:

```
#tcpdump "tcp and (tcp[13] & 0x02 != 0) and
(tcp[13] & 0x10 = 0)and (not dst port 53) and
   (not dst port 80)
and (not dst port 25) and (not dst port 21)"
```

As you can see from these examples, tcpdump's filters also allow the use of parentheses and logical operators. But what if you decide to implement a more complex filter? We've borrowed an example for **Listing B** from *Intrusion Detection: SHADOW Style*, which is referenced at the end of this article. This is a bad-events filter, which will detect any packets that indicate suspicious activity warranting further attention.

Obviously, it isn't reasonable to type such complicated filters on the command line. You can save such filters in a file and then read in by tcpdump at runtime with the `-F` flag:

```
#tcpdump -F filterfile
```

To understand what the last filter does, you have to master tcpdump and network protocols. Read the man pages for tcpdump(1). There are also a couple of excellent books about network protocols for further reading, which you'll find listed at the end of this article.

What to do with tcpdump?

First of all, tcpdump is a great utility for studying network protocols. For instance, **Listing C** shows how you can use tcpdump to see and analyze the setup phase of a TCP connection, which is generally referred to as a three-way handshake.

You'll find that tcpdump is an excellent network analyzing, troubleshooting, and debugging tool. You can use it for troubleshooting and analyzing TCP, DNS, NFS, PPP, RIP, AppleTalk, and other network protocols. For example, to troubleshoot network problems, dump all activity occurring on an interface:

```
#tcpdump -i hme0 -vv
```

For RIP troubleshooting, you can use the following filter:

Listing B: *A bad-events filter that can detect potential network threats*

```
#tcpdump "(tcp and (tcp[13] & 3 != 0) and ((dst port 143) or (dst port 111) or
(tcp[13] & 3 != 0 and tcp[13] & 0x10 = 0 and dst net 172.16 and dst port 1080) or (dst port 512 or dst
port 513 or dst port 514) or
((ip[19] = 0xff) and not (net 172.16/16 or net 192.168/16)) or
(ip[12:4] = ip[16:4]))) or (not tcp and igrp and not dst port 520 and ((dst port 111) or (udp port 2049)
or ((ip[19] = 0xff) and not (net 172.16/16 or net 192.168/16)) or (ip[12:4] = ip[16:4])))"
```

Listing C: *Use the output from tcpdump to study how network protocols work*

```
#tcpdump host pine.tree.com and host oak.tree.com and port telnet
tcpdump: listening on hme0
08:16:48.519576 pine.tree.com.33102 > oak.tree.com.telnet: S 2970775184:2
970775184(0) win 8760 <mss 1460> (DF)
08:16:48.519602 oak.tree.com.telnet > pine.tree.com.33102: S 3013029286:3
013029286(0) ack 2970775185 win 8760 <mss 1460> (DF)
08:16:48.520242 pine.tree.com.33102 > oak.tree.com.telnet: . ack 1 win 8760 (DF)
08:16:48.522879 pine.tree.com.33102 > oak.tree.com.telnet: P 1:28(27) ack 1 win 8760 (DF)
08:16:48.522913 oak.tree.com.telnet > pine.tree.com.33102: . ack 28 win 8760 (DF)
08:16:48.540494 oak.tree.com.telnet > pine.tree.com.33102: P 1:16(15) ack 28 win 8760 (DF)
^C
```

```
#tcpdump -i hme0 -s 1024 port routed
```

You can also use `tcpdump` to create a powerful, network-based Intrusion Detection System (IDS). SHADOW, which was designed by the Naval Surface Warfare Center, Dahlgren Division, is one of the first freely available toolkits based on tcpdump for detecting intruders. *Intrusion Detection: SHADOW Style* will guide you step by step in building your own very robust network-based IDS. You will also find many useful tcpdump filters in this book.

Obviously, you can run tcpdump on your hosts to detect network-based attacks, even without building your own IDS. For example, to detect teardrop attacks, you may use the following filter:

```
#tcpdump "udp and (ip[6:1] & 0x20 != 0)"
```

To detect IP packets where the source and destination addresses are equal (classic land attack), you can use:

```
#tcpdump "ip[12:4] = ip[16:4]"
```

If you would like to understand more about packet filtering based on tcpdump and network attacks, the SANS Institute (**www.sansstore.org**) offers excellent courses on this topic.

Conclusion

It's always a good idea to *sniff* your own networks for any suspicious behavior. Based on our experience, tcpdump is the right tool for the job. Working with tcpdump will also allow you to understand in detail the various network protocols.

If you periodically sniff your networks, you'll be able to find the network bottleneck. You'll also be able to identify bad-guy network attacks by creating your own tcpdump-based IDS. ✳

Further reading

- *Intrusion Detection: SHADOW Style.* Stephen Northcutt and the Intrusion Detection Team. SANS Institute (**www.sansstore.org**).
- *TCP/IP Illustrated, Volume I.* Richard Stevens.
- *Internetworking with TCP/IP, Volume I.* Douglas E. Comer.

Distributed computing with CORBA and Java

DOWNLOAD ftp.elementkjournals.com/sun/sep00

by Paul A. Watters

Distributed systems is one of the fastest growing areas in enterprise computing today. As the processing requirements for modern applications can span continents, let alone the globe, there has been a huge push to enable cross-platform, language-independent methods for exchanging data around the network. The Internet has also introduced new application domains associated with business-to-business e-commerce, where defined operations must be performed between sometimes distrusted hosts. Identification, authentication, and authorization clearly play a large role in allowing clients and servers owned by different organizations to communicate and exchange data safely and effectively.

One method that has been developed to facilitate this exchange of data is CORBA, or the Common Object Request Broker Architecture, developed by the Object Management Group (OMG). *CORBA* is a cross-platform, language-independent method of specifying and implementing interfaces between two Object Request Brokers (ORBs). Although many C developers are familiar with Sun's RPC (Remote Procedure Call) software libraries, RPC was clearly focused on client/server operations, whereas CORBA applications feature ORB-ORB transactions, which can be client/server or server/server in nature. ORBs communicate with each other using the standard Internet Inter-ORB Protocol (IIOP), which is implemented as a UDP service.

This feature makes CORBA particularly useful for implementing many applications that involve distributed financial transactions, for example, which require distributed replication and data element validation. The other major difference between RPC and CORBA is that CORBA is object-oriented; it passes objects by reference between ORBs, so that remote ORBs can update data where necessary, without relying on a return data type. This feature overcomes some of the limitations in Java, since methods are invoked with parameters passed by value.

Since Java is now one of the dominant languages for distributed computing, it's no surprise that Java middleware developers have increasingly favored CORBA over Java-specific remote invocation methods, such as RMI. In fact, RMI can even be implemented over IIOP, demonstrating the flexibility of CORBA services. Java and C++ developers can also now easily write clients and servers that can communicate with each other without the need for non-standard, third-party libraries.

Choosing an ORB

If you're a Solaris application developer, you may have read that developing applications using CORBA is difficult. Alternatively, as an administrator, you may be called upon to support CORBA applications, without a clear explanation of the essential components of ORBs and client and server components. Although the OMG has released the CORBA API and some documentation through their Web site at **www.omg.org/**, there's still a gap between specification and implementation that isn't always well explained in literature.

In this article, we'll develop a simple application that verifies a password and a username. This simple process is found in virtually all distributed applications that require authentication. We'll also examine the deployment of the client, the server, and the actual class and method implementation required to support the operation. The code for the example is included in its entirety, meaning that you can type it in and try it out! Or, if you prefer, download it from our ftp site.

There are many ORBs currently available on the market for Solaris, including freeware, reference implementations, and several high-quality commercial implementations. In this article, we'll deploy both the client and server ORBs using the VisiBroker ORB from Inprise Corporation (**www.inprise.com/**).

One of the advantages of VisiBroker is that it's a complete development and deployment package, providing support for developing both Java and C++ client and server applications on Solaris. In addition, the ORB integrates completely with other enterprise-level services, such as Enterprise Java Bean (EJB) support and servlets. Thus, it's feasible to deploy any *n*-tier-distributed application using a single package, which reduces maintenance, overhead, and support costs.

Developing a distributed Java application

Eight steps are involved in developing and deploying a CORBA application with Java. First, specify the required modules and interfaces by using the Interface Definition Language (IDL), which is similar to C++. This includes parameter definitions for methods that must be passed from client to server (`in`), from server to client (`out`), and in both directions (`inout`). A special wrapper class is required to pass simple types (such as `ints`), whereas objects can be specified directly.

Second, generate the server skeleton and client stubs from the IDL code by using the idl2java program supplied with VisiBroker. This generates a series of files which are responsible for the marshalling of objects between client and server at runtime, and which don't need to be modified by the developer or deployer.

Next, implement all of the modules, interfaces, classes and methods defined in IDL using Java. Create a server in Java that initializes a server ORB and defines its runtime characteristics.

Now, create a client in Java that initializes a client ORB, contains the appropriate pointer to a server ORB, and invokes methods on the remote server. Then, compile all the class files for the server, client, and implementation using vbjc. Finally, start the server using vbj and execute the client using vbj.

A user authentication sample

Let's discuss each step in detail. First, the IDL code is used to define the interface between the client and server. In fact, it's used to generate client stubs and a server skeleton from which the client and server can be implemented in Java. This saves time and reduces errors due to coding and type differences. In addition, you could use the same IDL to develop a C++ server skeleton and a Java client stub, or vice versa, so it's quite flexible in operation.

Let's look at an interface definition (authenticate.idl) for a security module that performs username authentication. A single method is defined that returns a success or failure code from the server, and it passes two strings from the client to the server: a username and a password. These could be standard Solaris usernames and passwords, although we haven't specified an eight-character limit here for the username and password variables:

```
bash-2.03$ cat authenticate.idl
module security
{
   interface Authenticate
   {
         string checkUsername
         (
            in string username,
```

```
            in string password
        );

    };
};
```

That's all the code that's required on the IDL front. From this simple interface definition, we can generate a server skeleton and client stub for it, by using the idl2java command:

```
bash-2.03$ idl2java authenticate.idl
```

This command creates a subdirectory from the current directory that reflects the module name, in this case *security*. Thus, if we examine the contents of security, we should be able to see both the client stub and server skeleton, if there were no idl2java syntax errors:

```
bash-2.03$ ls security
_AuthenticateStub.java      AuthenticateOperations.java
Authenticate.java           AuthenticatePOA.java
AuthenticateHelper.java     AuthenticatePOATie.java
AuthenticateHolder.java
```

As you can see, each of the associated Java source files has the name of the interface prefixed (`Authenticate`), followed by a specific function (e.g., `AuthenticateHolder` defines any special holder objects required to wrap data types which are declared `out` or `inout`). The developer and deployer rarely have to edit these files, so it's best to focus on implementing the interface, classes, and methods defined in IDL.

With the single interface defined, we must create a Java source file called `AuthenticateImpl.java`, which simply indicates that the files contain the implementation of the Authenticate interface. Further, we must define a single method `checkUsername`, in accordance with our interface specification, in the class `AuthenticateImpl`.

In this example, we've defined two private variables that contain the target username and password (`president` and `lincoln`), but in a real application, these would be retrieved from a relational database by using the Java Database Connectivity Classes (JDBC), or EJBs. Here, if the client's username and password match those stored on the server, then a `PASSWD_OK` string is returned to the client, or a `BAD_PASSWD` string is returned. Obviously, the client should interpret these strings in a sensible way. **Listing A** shows the code.

The next step involves writing a server, which initializes the ORB on the server side. The key definition here is the Portable Object Adaptor (POA; in this case `security_agent_poa`), through which objects are activated on the server. Thus, the client that we develop later will also need to specify which POA it intends to connect to. Let's examine a standard server for our application in **Listing B**.

After implementing the server, the client is straightforward; we initialize a client ORB, identify

Listing A: *Our client authentication implementation code*

```
bash-2.03$ cat AuthenticateImpl.java
public class AuthenticateImpl extends security.AuthenticatePOA
{
   private String _username="president";
   private String _password="lincoln";

   public String checkUsername(String username, String password)
   {
if (username.equals(_username) && password.equals(_password))
      {
         return "PASSWD_OK";
      }
         else
      {
         return "BAD_PASSWD";
      }
   }
}
```

Listing B: *The server process that handles authentication*

```
bash-2.03$ cat Server.java
import org.omg.PortableServer.*;

public class Server {
   public static void main(String[] args) {
   try {
      org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init(args,null);
      POA rootPOA =
         POAHelper.narrow(orb.resolve_initial_references("RootPOA"));
      org.omg.CORBA.Policy[] policies = {
      rootPOA.create_lifespan_policy(LifespanPolicyValue.PERSISTENT)
      };
   POA myPOA = rootPOA.create_POA( "security_agent_poa",
      rootPOA.the_POAManager(), policies );
   AuthenticateImpl securityServant = new AuthenticateImpl();
   byte[] securityId = "Authenticate".getBytes();
   myPOA.activate_object_with_id(securityId, securityServant);
   rootPOA.the_POAManager().activate();
   System.out.println(myPOA.servant_to_reference(securityServant) +
   " is now available.");
   orb.run();
      }
      catch (Exception e) {
      e.printStackTrace();
      }
   }
}
```

the name of the server POA (`security_ agent_poa`), and pass the parameters for the remote method that will be invoked (`checkUsername(username,password)`). Obviously, we have to define `username` and `password` before invoking the remote method. In this client, we interpret the returned string as either indicating that the user's credentials were verified or were not verified, and print an appropriate message. **Listing C** shows the code.

Listing C: *Our client that will access the server via CORBA*

```
bash-2.03$ cat Client.java
public class Client
{
   public static void main(String[] args)
   {
      String message;
      String username="president";
      String password="lincoln";
      try
         {
         org.omg.CORBA.ORB orb =
            org.omg.CORBA.ORB.init(args,null);
         byte[] securityId = "Authenticate".getBytes();
         security.Authenticate a =
            security.AuthenticateHelper.bind(orb,
         "/security_agent_poa", securityId);
         message=a.checkUsername(username,password);
         if (message.equalsIgnoreCase("PASSWD_OK"))
         {
         System.out.println("credentials authenticated");
         }
         else
         {
            System.out.println("unable to authenticate
            ➥credentials");
         }
      }
      catch (Exception ex)
         {
            System.err.println(ex);
         }
      }
}
```

Deploying a distributed Java application

After creating the client, server, and implementation source in Java, it's then time to compile everything by using the `vbjc` command (VisiBroker for Java Compiler). In our case, we can type

```
bash-2.03$ vbjc *.java security/*.java
```

and all of the appropriate class files should be generated as shown in **Listing D.**

Start the server

If all of these steps have succeeded, then you can attempt to start the server by using the `vbj` command as follows:

```
bash-2.03$ vbj Server
Stub[repository_id=IDL:security/
Authenticate:1.0,key=ServiceId
[service=/security_agent_poa,id=
{12 bytes: [A][u][t][h][e][n][t]
[i][c][a][t][e]}]] is ready.
```

The module security and the interface `Authenticate` are now available on the server! Next, we can attempt to execute the client, also by using the `vbj` command. If the server isn't responding, you'll get an exception:

```
bash-2.03$ vbj Client
org.omg.CORBA.OBJECT_NOT_EXIST:
  Could not locate the following POA:
        poa name : /security_agent_poa
  minor code: 0  completed: No
```

If the server is running, you should see the response:

```
bash-2.03$ vbj Client
credentials authenticated
```

Of course, if you change the `password` in Client.java to be `clinton`, you'll have the error message printed to the display.

Summary

In this article, we have examined a very simple implementation of an authentication module in

Java, as specified in CORBA IDL. Although the development and deployment stages involve eight discrete steps, the ability to pass objects by reference between networked hosts is incredibly important to distributed applications. Of course, we have only examined interfaces consisting of a single interface and a single method defined for a module; a real-world module may contain several interfaces, each with multiple methods. Although it's not currently possible to overload methods within a particular interface, CORBA, in general, is particularly well suited to object-oriented application development on the enterprise Solaris platform. ✳

Listing D: *All the files needed to implement our CORBA example*

```
bash-2.03$ ls
authenticate.idl*       Client.class          Server.class
Client.java             Server.java           AuthenticateImpl.class
AuthenticateImpl.java   security/

bash-2.03$ ls security
_AuthenticateStub.class         AuthenticateHolder.java
_AuthenticateStub.java          AuthenticateOperations.class
Authenticate.class              AuthenticateOperations.java
Authenticate.java               AuthenticatePOA.class
AuthenticateHelper.class        AuthenticatePOA.java
AuthenticateHelper.java         AuthenticatePOATie.class
AuthenticateHolder.class        AuthenticatePOATie.java
```

Further reading

The *OMG IDL/Java Language Mapping Specification* describes the operations that must be performed by a CORBA implementation in generating client stubs and server skeletons from IDL. This information can be very useful if you're trying to troubleshoot a specific implementation. You'll find this specification at the OMG Web site at **www.omg.org/**. A FAQ document about VisiBroker is also available from the Inprise Web site at **www.borland.com/visibroker**.

About our contributors

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

Boris Loza holds a Ph.D. in computer science from Russia. He worked as a UNIX administrator and developer for 10 years. Currently he is working for Fidelity Investments Canada in the position of data security and capacity planner, doing IT security for UNIX, Windows NT and Novell. He has a daughter Anna and enjoys reading computer and mystery books and watching movies. He can be reached at **Boris.Loza@FMR.com**.

Alan Orndorff has been working with computers since 1990. He is using Solaris as a platform for Lotus Notes and at home in his spare time. He currently lives in San Francisco and can be reached at **dwarf333@hotmail.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™
Tips & Techniques for users of Sun Solaris

Inside Solaris (ISSN 1081-3314) is published monthly by Element K Journals, a division of Element K Press, 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

Element K Journals Customer Relations
500 Canal View Boulevard
Rochester, NY 14623

Or contact Customer Relations via Internet email at journals@element-k.com.

Editorial

Editor.....Garrett Suhm

Assistant Editor .....Jill Suhm
Managing Editor .....Michelle Rogers
Copy Editors.....Rachel Krayer
Glenna Lechner
Contributing Editors .....Clayton E. Crooks II
Boris Loza
Alan Orndorff
Paul A. Watters
Print Designer .....Rachel J. King
Cover and Content Design.....Melissa Ribaudo

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) 272-0064

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

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

Element K Journals

General Manager ..... Kelly Baptiste
Manager of Customer Relations ..... Nicole Pate
Manager of Operations ..... Cristal Haygood
Manager of Print Design ..... Ian Caspersson
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

Printed in the USA.

Price

Domestic .....$129/yr ($11.00 each)
Outside US .....$149/yr ($13.00 each)

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

Back Issues

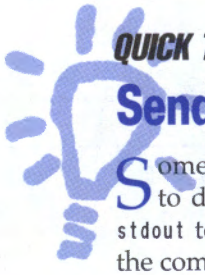
To order a back issue from the last six months, call Customer Relations at (800) 223-8720. Back issues cost $11.00 each, $13.00 outside the US. You can pay with MasterCard, VISA, Discover, or American Express.

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@element-k.com.

Coming up...

- Setting up POP3
- Using ToolTalk



QUICK TIPS

Send output and errors where you want

Sometimes when problems occur, you'll want to direct the normal output of the command `stdout` to one location, and the error output from the command `stderr` to another location.

As an example, suppose you're running an unattended, custom, tape-backup utility at night. Normally, you want to write the standard output to one file, while writing the backup errors to a separate file. You can easily accomplish that with the following:

```
tape-backup > tape.out 2> tape.errs
```

Other times, usually when you're working interactively, you want to redirect both the standard output stream and the standard error stream to the same location. A good example of this is when you're first creating a shell program and it's generating a lot of errors in its output. In debug mode, you may want to see the normal output of the program and the debug output in the same data stream. You could type this:

```
myscript | more
```

If you have a lot of output, the errors will go right off your screen. Here's how to send the standard output stream and standard error stream into the `more` command together:

```
myscript 2>&1 | more
```

In this case, the normal output and error output are both controlled by `more`, which is just what you want.

Use type to find commands and learn more about them

When you type commands like `cd`, `ls`, `whence`, `pwd`, and others, do you wonder where they come from? Using the Korn shell's type command, it's easy to find out.

The `type` command is a great tool for learning more about your system, and more about where commands are coming from. A simple command like the following:

```
type cd pwd ls who whence
```

Will yield the following results:

```
cd is a shell builtin
pwd is a shell builtin
ls is a tracked alias for /bin/ls
who is a tracked alias for /bin/who
whence is a shell builtin
```

This tells you that the `cd`, `pwd`, and `whence` commands are built into the Korn shell, while `ls` and `who` are commands that exist in the /bin directory. This can help you when you're trying to understand where a program is coming from and why it may or may not be working.

Why making copies of UNIX CDs doesn't work

Did you just buy a shiny new CD burner for your PC and then try to make a copy of a UNIX CD? It doesn't work since Windows (both 98 and NT) doesn't support ISO 9660 with Rock Ridge extentions that are necessary to handle UNIX file systems. Simply point your browser to:

ftp://tsx-11.mit.edu/pub/linux/packages/mkisofs/

You can retrieve the source for mkisofs here. This program allows you to make an image of a UNIX CD that can be read by commercial PC CD burning packages such as Adaptec's Easy CD Creator. ✳