by Peter Collinson, Hillside Systems

STEPHEN SCHILDBACH

Faster... Faster...

Part of my personal millennium plan was the upgrading of my aging SPARCstation II. I've had a Sun Microsystems Inc. workstation for 10 years, starting with a Sun386i and migrating for one reason or another to the SPARCstation II. I also tried to keep up with operating system changes, spurred on to some extent by writing this column. However, Solaris 2.6 was nearly more than a SPARCstation II could handle. It was usable, but slow in spots. Those spots were mostly interpreted systems like Java or Tcl/Tk. I also had the feeling that its ability to handle Ethernet traffic was not keeping pace with the other machines on the site, which are all Pentiums of varying antiquities. The Solaris 2.6 system on my SPARCstation II was left to rot. Upgrading it to ensure that it was Y2K-compliant seemed to be a pain, especially since the CD drive was an original "single speed" device inherited from some older system that I owned at some point in the past 10 years.

I decided to buy a new machine, something that you should think about quite hard when it's your own money. However, I can't say I put a great deal of thought into what choices were really available to me. I'm used to having a SPARC and wanted to continue. I decided to upgrade to an Ultra 10. Some of my reasons simply came down to engineering. I hoped that I could move my personal working environment with no pain from the old system to the new.

Another issue is support. I'm used to being supported, and have availed myself of hardware support several times in the past 10 years. When the machine breaks, which is rare but deeply annoying, I need to have someone rush out to fix it. Also, if you pay for maintenance, Sun's software support is worth having, if only for the steady stream of system patches it emits.

Working out what options I had for a new machine turned out to be much harder than I thought. Sun seems to be geared up to shift boxes to uncomplaining large companies whose purchasing departments probably blame their local systems administrators if several new systems arrive with problems. I was determined to go into this with my eyes open and kept asking questions that the diverse range of perfectly charming people at the other end of the phone could not answer. I was left to piece together my own set of possibilities, rather than having someone give me the answers I needed. I'll omit the full details to spare Sun's blushes.

I ended up buying a 400-MHz Ultra 10 with 256 MB of memory and 18 GB of disk. I decided on the latter because I also bought a SunPCi card, which is a coprocessor that allows you to run Windows 95 or NT 4.0 on a machine that is a "real" processor, but has no peripherals. It takes input from your keyboard or mouse and displays output in a window on the X11 server on the host. The card uses the Sun disks to store disk images, and because these were likely to be big, I thought that extra disk would be a good idea. A wise thought, it turns out.

When the new system arrived, the big change for me is its speed of operation. The machine is considerably faster than the SPARCstation II. As a nontouch typist, whose inaccuracy in typing means the most used key on my keyboard is Delete, speed makes a huge difference. My typing is constrained by seeing the characters being echoed back to me to confirm that my hands are in the correct place and I'm hitting the correct keys. I am now creating text much more speedily than before, although choosing the correct words takes the same amount of time.

My initial fears that I would hate Sun's nonoptical mouse have proved groundless. The roller-ball mouse is light and easy to use. It comes with a soft nonshiny mouse pad, and that might be the reason for its success. Actually, another design win with the mouse is that it plugs directly into the keyboard and there is only one cable between the keyboard and the tower box on the floor. Of course, Sun's been doing this for aeons, but good, thoughtful design is always worth mentioning.

The machine's high-quality graphics allow me to have a larger screen area in terms of pixels, and this is a win. It would be nice if Sun gave us some three-dimensional system applications to show this off–how about a 3D tree viewer for the file system?

Solaris 7

On the surface, Solaris 7 isn't a great leap forward for the general user. This is a good thing because great leaps tend to break old software. I'd kept my Sun version of `sendmail` in line (or nearly so) with those versions being made available by Sendmail Inc. Creating the `sendmail` setup for the machine was easy: just take the old control file in `m4` and type `make`. I then realized that Sun's `sendmail` now supports the BSD "hash" format files, and I could change some of the older setup I used previously.

Installing the system and placing it on the Internet was easy. The only real gotcha on Solaris 7 is the Domain Name System (DNS), which uses the more recent control file format.

Whenever you move from one system to another, problems are created by the local changes you've made. Over the years, I'd tried hard to keep all my changes in `/usr/local`. Where I could not do this, because I was tailoring some standard system file, I left a marker. Whenever I alter a system configuration file, called say `fredrc`, I'll retain the original one in `fredrc.orig`. One reason for this is to allow immediate backtracking, the other is that you can scan the file system to find what you've altered. Before hacking on the new machine, it's a good idea to ensure that any change you made originally is still required and that there isn't some other new way of achieving the same end.

If you keep all your local commands in `/usr/local/`, then this whole tree will hopefully port intact into the new system. When I set up my new machine, I crossed my fingers and simply copied my old `/usr/local` from Solaris 2.6. To my considerable relief, programs have mostly worked. I didn't want to do massive recompilation of my personal tool set on day one.

There are some funnies. I use Brent Welch's `exmh` as a mail user agent and this depends on Tcl/Tk. Initially, the program didn't appear to work as expected and I blamed the Tcl/Tk setup. I found a precompiled Solaris 7 pair of packages on the Scriptics Corp. Web site and installed them. The new binaries didn't seem to make much difference. Then I realized that the dynamic link loader wasn't looking in the right place, so fronting the various `exmh` user programs with shell scripts that set `LD_LIBRARY_PATH` cured the problem. I also suspect the way mailboxes are locked on Solaris 7 has changed from the method used by Solaris 2.6. I haven't had a chance to check this yet.

I have also found that the BSD `locate` program suite doesn't seem to work as expected. I wrote about this program in my article, "Moving to Solaris" (*SunExpert*, May 1996, Page 20). The idea of the program is that you build a searchable list of files on your system every day (or so). Building the list is easy, you just do a `find` to locate every file in the file system and store the list. The `locate` system has a couple of magic programs that build a hash table from this list and a command called `locate` that accesses the hash table to search and print file names.

The search functionality was included in the `find` command on the BSD system releases, so you could say

```
$ find .html
```

to find all the files that contained the string `.html`. However, POSIX didn't bless the functionality in the `find` command and it was revived in a new non-POSIX command called `locate`. I find the action of the command invaluable, especially on new systems where I am always looking for files. My body keeps typing the old `find` command, so I have created a shell function:

```
function find
{  if [ $# -eq 1 ]
   then
       /usr/local/bin/locate $1
   else
       /usr/bin/find "$@"
   fi
}
```

This allows me to invoke the command using the old syntax.

For some as yet undiscovered reason on Solaris 7, either the hash table is built incorrectly or the `locate` program doesn't work. Either way, when I type `find` to invoke my shell function or `locate` to call the command directly, I am presented with garbled file names. However, the machine is now fast enough that I've replaced the lookup program by a call to `grep` that simply scans through the list of files that are recreated every night. I've added the `-i` option to `grep` to make the general search string case-independent.

I really have had no other problems. When my Sun C compiler arrives (another story I won't bother you with), I plan to slowly migrate from binaries compiled on the old system to binaries that are compiled on the new. I am unsure that I will need the 64-bit functionality of the new system for any of my programs, however.

Good Things

I said that Solaris 7 doesn't present many changes for the user. This is somewhat unfair. There are changes, but they are generally subtle. For example, the manual pages are now distributed in

Standard Generalized Markup Language (SGML) and not `troff`. The `troff` versions can be compiled from the SGML.

The SGML format (or "DTD," which is the three-letter acronym for such things) is SolBook, a subset of the DocBook DTD. The benefit here is that the Answerbook system understands SGML, and the manual pages are now accessible directly from that system. The `man` command has been altered to understand both formats, so there is, or should be, no problem installing traditional manual pages. I have just today started leafing through a new book on DocBook from O'Reilly & Associates Inc. (see below for more information).

The Common Desktop Environment (CDE) has, I suppose, "progressed." It seems much more resilient than the previous version. When you changed things in the previous version, it was often prudent to log out and in again to reset the various dynamically created control files; the new version seems to permit alterations on the fly with no adverse effects. There are some new features. The first thing you'll notice is that the Front Panel has acquired color and dynamic movement. I went with the flow and moved everything from my old home directory to the new, *except* the old `.dt` directory. The `\*C.dt\fP` directory is the configuration and working area for CDE. By not taking my old one, I started the new CDE from scratch.

Many of the bugs in CDE that I mentioned in my CDE articles (see "Common Desktop Environment," *SunExpert*, June 1996, Page 22, and "Customizing CDE," June 1997, Page 30, `http://sw.expert.com/C2/SE.C2.JUN.97.pdf`) have been fixed, and the resulting product is considerably more usable than before. For example, if you move about the file system in the file manager and open a terminal, the working directory of the terminal is set to the place you are looking at in the file manager window. One bug has not been fixed: I want to be able to tell the various page imaging programs that my pages are designed to work on A4 and not letter-size paper. The fact that this has not been changed is not surprising because an American engineer once pronounced it "not a bug."

There are a bunch of new things in CDE that make point and click somewhat more of a reality. You can create a file in a bookmarks directory (which lives in `~/.dt/bookmarks`) that can be clicked and passed as a URL into the running version of Netscape. These files contain the URL of the target, and can be created by dragging some text that is a URL into the bookmarks directory. The directory is "magic." Here, you can create files that contain a URL, which will turn into an action that when clicked loads Netscape and takes the user to the specified site.

You can create "cards" that again are stored in a private directory in `~/.dt`. These form an electronic address book, so you can select a name (whether it be a person, a machine or a Web page), click and be directed to the target address using the appropriate technology–email, `rlogin` or Netscape. The address book format is based on the Vcard standard that's used by some palmtops to act as electronic contact cards.

I still have a niggle that CDE tools tend to plonk things into your home directory. For example, if you create an action using the Action Editor, it creates the working file in `~/.dt/types`. However, it places the "action button"–the file you are supposed to click on–in your home directory. You can move this to somewhere that's hidden away and things still work. Actually, I've created a directory `~/.dt/appmanager/personal` to hold these buttons. This directory pops up in the Application Manager window as a folder named `personal` and is exactly the right place to store private actions. However, if I try to edit one of these actions, the Create Action GUI whinges about it not being created by itself. The trick is to find the actual action file in `~/.dt/types` and drop that onto the editor. However, when you save the file, it creates an action button on your home directory again. It seems safe to delete this and use the one you created earlier in the `~/.dt/appmanager/personal` directory. All this is confusing and is a mess that needs mending.

Working Out What's Happening

With any new system, I seem to spend time working out what's happening and trying to impose the way I want to work onto the ideas that the system designers *thought* I might like to work. The basic sources of information are the Answerbooks and the `man` pages. Although, as I've noted before in these pages, there are several commands that you can see on a `ps` listing that simply are not documented at all. CDE is notorious for having undocumented commands and, on some occasions, inspired guesswork has to be used to find out what the commands do and why they are failing.

My usual sequence of actions when I am trying to understand what is happening is to first type the `man` command. When I find nothing, I revert to the Answerbooks and search for the string. If that fails to yield anything, then maybe a `find` (or `locate`) for the file name will show you that there's a text file, or a manual page that's not showing up in the `man` command. If that fails, then you need to investigate the binary. Point the `strings` command at it; sometimes, there's a help message or some other form of identification that may help you deduce where the program fits into the world.

When everything else fails, but you think running the command is an option (sometimes it isn't, so be careful), then the `truss` program is your friend. The job of `truss` is to print out all the system calls that a particular process has made. Remember that a UNIX process talks to the outside world via system calls, so the ability to print what system calls have been executed and their parameters can tell you quite a bit about what a process is or isn't doing. There are two useful command-line forms:

```
$ truss command
```

This runs a command and prints all the system calls it makes. To try this, say

```
$ truss ls .
```

and `truss` will print quite a small amount of information. It prints the information to the error channel, so if you want to capture all the output to a file, you'll need to say

```
$ truss -o file ls .
```

If you now examine that file, you'll find that the first part of the program worries about loading the runtime libraries into the process address space. You can see several lines like the following:

```
open("/usr/lib/libc.so.1", O_RDONLY)   = 4
fstat(4, 0xFFBEECD4)                   = 0
mmap(....
```

The sequence is the dynamic loader creating the remaining parts of the process by mapping parts of the library into its memory. You need to skip past all these lines to find the work that the process is doing. For `ls`, the work starts here

```
time()                          = 945189387
ioctl(1, TCGETA, 0xFFBEF14C)    = 0
```

The program establishes the time, and then looks at its standard output channel using the `ioctl` system call. Actually, it's trying to find out if its standard output channel is a terminal; this call will fail if the standard output is a regular file. Remember that `ls` gives you output in columns when the standard output is a terminal. The program continues

```
brk(0x000269C8)                 = 0
brk(0x0002E9C8)                 = 0
```

The `brk` system call obtains some memory for the process to work in. Now things get interesting.

```
lstat64(".", 0xFFBEF0C8)        = 0
open64(".", O_RDONLY|O_NDELAY)  = 3
fcntl(3, F_SETFD, 0x00000001)   = 0
fstat64(3, 0xFFBEF008)          = 0
```

The `64`s here are all special system calls that take 64-bit arguments rather than 32-bit ones. The `lstat64` call is the same as the `fstat` call, except it's given the name of a file for which to return details and `lstat` doesn't follow symbolic links, so I guess it's checking for symbolic links. It decided that it doesn't have one and opens the file, which is a directory. It then sets the "Close on Exec" bit in the file descriptor using `fcntl`. I'll guess that the code uses a standard directory reading routine that does this and the `fstat`. The program finishes

```
brk(0x0002E9C8)                    = 0
brk(0x000309C8)                    = 0
getdents64(3, 0x0002E6D8, 1048)    = 176
getdents64(3, 0x0002E6D8, 1048)    = 0
close(3)                           = 0
ioctl(1, TCGETA, 0xFFBED1E4)       = 0
write(1, " # c d e 3 . m s #     c".., 31) = 31
llseek(0, 0, SEEK_CUR)             = 4379
_exit(0)
```

It reads the directory contents and prints the data.

I feel it's worth going through the output of a simple command in some detail to show you what you can deduce from looking at the output. It's rare to get into this level of detail. Mostly, I use `truss` to find what files a process is opening.

There are many options for `truss`, and you should read the manual page. Here are a couple that I use more often than not.

The -p option is followed by a process ID. It will trace the running process, assuming that you have appropriate ownership rights. The switch is useful when you have a process that's running away and you want to find out why. I tend not to just kill a runaway because what has happened will certainly happen again, and by killing it you are destroying the evidence. You may not be fixing the problem. Beware that `truss` will slow down the process that it's tracing so that it can print the information about each system call. So using `truss` to trace processes that are dependent on time can have unwanted effects.

Normally, `truss` will only report on one process, and if that process forks to produce a child, then the information in which you are interested may not be available. The -f option allows `truss` to track any children of the process that you are tracing, and this can be very helpful when looking at daemon processes that do their work by generating a new process. This was how I originally figured out how Solaris 2.6 locked mailbox files.

Solaris 7 provides some new functionality for `truss` that can be used to remove some of the guesswork that I have entered into above. I should say that I wrote the guesswork text before using this feature. The -u option is followed by a shared library name, or a comma-separated list of shared library names, and will show the entry and exit from each of the routines that are invoked. Here's the bit at the start of the `ls` program:

```
-> libc:time(0x0, 0xff3338f0, 0x0, 0xff33723c)
time()                              = 945192671
<- libc:time() = 0x38567edf
-> libc:isatty(0x1, 0xff3338f0, 0x0, 0xff33723c)
ioctl(1, TCGETA, 0xFFBEF154)        = 0
<- libc:isatty() = 1
-> libc:getopt(0x1, 0xffbef22c, 0x13c40, 0x0)
```

It shows that the program called the `time` routine and then `isatty` to work out if it was talking to a terminal. You may recall I deduced the call to `isatty` by knowing how it worked. As I said, you'll probably never need to get to this level of detail with `truss`, but it is one tool that can help you investigate the mysteries of what is happening on your machine.

Further Investigation

The Scriptics site from which I obtained my precompiled Solaris 7 binaries of Tcl/Tk is `http://www.scriptics.com`. The new book on the DocBook SGML DTD is *DocBook, The Definitive Guide* by Norman Walsh and Leonard Muellner (published by O'Reilly & Associates, 1999, ISBN 1-56592-580-7).

***Peter Collinson** runs his own UNIX consultancy, dedicated to earning enough money to allow him to pursue his own interests: doing whatever, whenever, wherever… He writes, teaches, consults and programs using Solaris running on an UltraSPARC/10. Email:* pc@cpg.com.