by Peter Collinson, Hillside Systems

MICHELLE FRIESENHAHN WILBY

Who's Doing What, When, Where?

UNIX was designed to be used in a cooperative environment, so it's natural that it supplies programs that help you to know things about the other users on the machine, if only to identify that idiot who is gobbling up resources. UNIX has always been a system where everyone can see what is happening on the machine and has several tools aimed at providing system and user information. Many of the system tools were originally designed to provide debugging information about resources, but can now be used to investigate the state of the machine, telling you what that idiot is doing.

Perhaps the earliest tool aimed at supplying information about users was the `who` command. The `who` command tells you who is logged onto the machine, which terminal they are using and the time they logged on. These days, logins are mostly done over the network, and the `who` command also prints the name of the machine from which the network connection was made.

The `who` command simply prints data from a file. Traditionally, the file is `/etc/utmp`, but I notice on Solaris 2.5 that things have become more complicated. I'll get to that later. I'll talk about SunOS first. It's a simpler system and closer to the original.

Basics of `who`

When you log into a machine, the login program writes a record in `/etc/utmp`. The record holds your login name, the name of the terminal line you used to log in, the name of the host that you logged in from and the time that all this happened. The record is fixed length and is written into a known position in the `/etc/utmp` file determined by the terminal name.

On SunOS, terminal access is controlled by the `/etc/ttytab` file. The file acts as a control file for the `init` program, which is responsible for starting up the program (`getty`) that sits on terminal lines waiting for users to log in. When the user logs out, their shell will die. The `init` process is notified and wakes up to start a new invocation of `getty`, listening on the line for a new user.

The `/etc/ttytab` file contains one line for every terminal attached to the system. The line position of a particular terminal in this file is its "slot," and this index is used to provide a unique position in the `/etc/utmp` file.

Incidentally, you'll find that you have to include all the pseudoterminals that you want to use on the system in this file. Their names are not needed for the benefit of `init` because they are mostly used by programs started from the network. However, when a pseudoterminal is used by the login program, it will expect to find a slot in `/etc/utmp`–the slot is provided by an entry in the `/etc/ttytab` file.

You need to be circumspect when editing the `/etc/ttytab` file on a live system. It's OK to change details of a line, perhaps turning a terminal line on or off, and it's also OK to add new terminal lines at the end of the file. But

make sure that you don't change the slot numbers of existing lines. Radical line changes to the file need to be done on a single-user system and followed by a system reboot to resynchronize terminal names to slots.

The entry in `/etc/utmp` that is written by the login program will need to be cleared when you log out. The login program cannot do the clearing because it's long gone. The login program used the `exec` system call to become your shell to allow you to work. One of the jobs of the `init` program is to notice that you have logged out and restart the program that listens for terminal connections on that port, so it's the `init` program that is responsible for clearing the entry for the slot in `/etc/utmp`.

Incidentally, the incorrect setting of the access permissions of the `/etc/utmp` file can supply an interesting security hole. It's sometimes set so that everyone can write to it. Setting global write permission is often done because there are some programs that legitimately wish to write to `/etc/utmp`, but you don't want to give those programs superuser status. Making the `/etc/utmp` file writable by all provides an unwelcome intruder with the ability to use a simple Romulan cloaking device. The intruder can create a tiny program that clears their entry in `/etc/utmp`. They are now cloaked and invisible to the `who` program, and this can be confusing.

Of course, the cloaking is imperfect; you can still see their processes using the `ps` command. I notice that the BSD/OS system places `/etc/utmp` in a special group (`utmp`), with the intention of using `setgid` to enable any program that wants to write to the file. Actually, I can find no program that is set to be `setgid` to the `utmp` group, so it's possible that access permissions on `/etc/utmp` can be established so that only root can write to it.

Solaris who

I said that Solaris is different. Actually, the differences are not fundamental: A record is still being written when you log in and cleared when you leave. The system on Solaris is an evolution of the original one. I suppose I should have said the *systems* on Solaris. There are two parallel `utmp` files on my Solaris 2.5 system: `/var/adm/utmp` and `/var/adm/utmpx`. The `/etc/utmp` file has become a symbolic link that points at `/var/adm/utmp`.

Some of the changes in the system are undoubtedly due to the changes in the `init` program that were made for System V. The `init` process is now a much more general process-spawning program, and its actions are logged into the `utmp` file. I suspect there are System V compatibility reasons that compel the need for two parallel files. This is the penalty for having defined binary file mapping onto a C structure that is pulled "raw" into a C program rather than having a well-defined program interface to read the data. I think that

having two files is fraught with danger.

The utmpx file format is different from utmp. The utmp file only contains the user name, the line that was used to log in and the login time. The utmpx file contains much more space for user names (32 characters rather than eight), and also holds a 256-character array to store the host name. It seems that the files are no longer managed using the terminal slot. They are variable-length files, and (I'm guessing) when the login program wants to store a new record it simply finds an empty slot in the file to write the data. The record contains a keyword that is used to relate the actual terminal with the data using a serial search through the file.

Incidentally, when I was looking at the who command, I noticed that it reads both the utmp and utmpx files. The program doesn't need to read both files because all the information in utmp is also present in utmpx. This bug was reported to Sun (in 1993, actually). The alternative version of who, found in /usr/xpg4/bin/who, just reads its information from the utmpx file. Thus, if you are worried about CPU or disk cycles (and you probably are not), then you can find a more efficient version of the who command.

Both utmp and utmpx have extra fields that store new information. I suspect that some of the new information is there to allow the who command to behave like the w command that originated on the Berkeley systems. The w command is useful because it not only tells you who is logged in but also indicates how long their terminal has been idle, the CPU seconds that the terminal has consumed and the current command being executed on the terminal. All this information can help you work out whether the person is actually sitting at their terminal and using the machine, or whether they left the terminal a while ago.

You can see all the new information stored in utmp and utmpx by using some options to the who command:

```
$ who -a
```

This gives you a complete dump. The output will include records that show the start-up of your system, and also empty slots in the file that give you information on past activity. A plus sign next to the terminal name shows active login records. The records that are written on the file are typed using a tag value, allowing various types of activity to be stored. For example, you can get the time that the machine was last booted using the -b flag:

```
$ who -b
       system boot  Jan 31 00:00
```

Giving who an illegal option like

```
$ who -x
```

will print a list of valid options and what they mean.

Login History

The who command only tells you the state of the machine at the moment that you run the command. To provide a history of activity on the machine, the login program appends the same record that it placed in the utmp file to the /var/adm/wtmp file. On Solaris, it also places the utmpx record at the end of the /var/adm/wtmpx file. The wtmpx file is supposed to track the wtmp file, but the file seemed to be garbled on at least two systems I looked at for this article.

You can dump the contents of these history files using the who command. On SunOS,

```
$ who wtmp
```

will just dump the file, giving you a listing of all the activity. On Solaris, the who command only prints login records, but you can use the -a flag to see most of the data. Actually, the best Solaris command to print all the data in the file is part of the accounting suite, fwtmp. Use the command

```
$ /usr/lib/acct/fwtmp < wtmp
```

You will have to look up some values in /usr/include/utmp.h to decode some of the tag values that are used to differentiate the records.

On SunOS, user data in the file will mostly come in pairs. You will see a login record containing a user name, a terminal line name and a time; and a logout record, where the user name is empty, the terminal line is present and, of course, the time is when the user logged out.

Records for users on Solaris come in threes: there's a "system" start-up record, written by the terminal monitor program or perhaps by telnetd, rlogind or another program that handles logins; a user start-up record that indicates that the user name of the person who has logged on; and a "dead process" record, written when the user logs off.

Both systems will have other records written into the file. If you change the clock using the date command, then there's a discontinuity in the time sequence on the machine, and the accounting programs need to know that the clock was reset. A pair of special time change records are written into the file by the date command. The first will give the

time before the change, and the second will give the time after the change. The difference between the two can be used to adjust the login times for people who were logged in when the time was reset. Actually, it's a good idea to avoid setting the date and time using the `date` command when the system is running in multiuser mode.

Also, on both systems, records will be written when the system is rebooted. The times associated with these administrative records can be used to give some idea of the availability of the system. If a clean shutdown is made, then a shutdown record is written. When the system reboots, a reboot record is written. So it's possible to deduce whether the system crashed by noting the absence of the shutdown record.

Actually, deriving system availability figures from the log can be a little hit and miss. The accounting information can give an incorrect picture. If the system crashes in the middle of the night when there has been no user activity, then the accounting programs will only see a reboot record written some time after the last active record on the file. The programs can have no real idea how long the system has been down. If someone is logged in when the system crashes, then they cannot attribute a correct connect time for that user, because it's not possible to know for certain when the system died.

On a clean shutdown, all the users will be logged out before the shutdown record is written. The reboot record will follow the shutdown record and its timestamp can be compared with that on the shutdown record to deduce how long the system was unavailable. However, the reboot record just tells us when the bootstrap happened, it doesn't tell us how much initialization time was used by the system. On Solaris, a record is written when the machine changes from one run state to another, so we can see when the system entered multiuser mode and was offering service to customers.

You'll find that many systems write records into the `wtmp` file (or files). Obviously, user-driven programs like `rlogin` or `telnet` do. Less obviously, you'll find records written by FTP or UUCP.

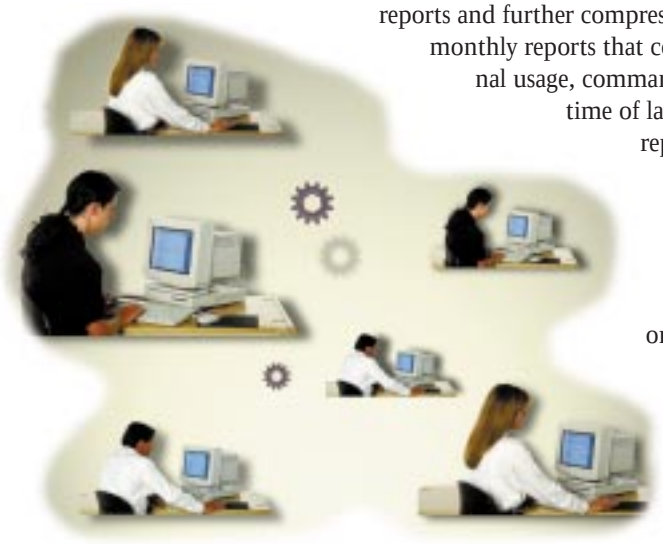
Process Accounting

The `wtmp` file can act as the basis for accounting for the connect time used by particular users on the machine. This perhaps had more relevance in the days before workstations when the user logged in once from their character-based terminal, and that single login was the sole source of their work. Now, users can log in several times to the same system, and connect time accounting is perhaps less relevant.

Both Solaris and SunOS support the logging of every process that has been run on the machine. The kernel has a module that is supplied with a file name and will write a record to the nominated file whenever a process terminates. By default on Sun systems, this file is `/usr/adm/pacct`. It's often called

acct on other systems. You should be aware that the file can grow to mighty proportions on a busy system and you must take the steps that the system provides to control the file size. By default, process accounting is turned off and needs to be enabled. Consult Sun's systems administration documentation.

The record that is stored for each process contains the information on the user who executed the command (the user and group ID); the controlling terminal for the command; the start time of the process; the user, system and elapsed time measured in clock ticks; the average memory usage and the bytes of I/O that the process performed; and a flag that indicates whether the process used superuser privilege or has forked a new process. Finally, the command name is stored as an eight-character string.



Mortals can look at the file with the lastcomm command, so you can find out the commands that someone else is executing. Commands are printed in reverse order of termination, so you can see the recent commands and use ^C to escape when you have seen enough. You cannot see the parameters to the commands. The lastcomm command may be useful when tracking bad guys on the machine, but the command name that is stored is simply the name of the file, and not its path. When the addictive rogue game first came out, we found that many people had private copies named for programs that they were supposed to be using, like vi. This meant that they could sit and play rogue, with the ps command telling the casual watcher that they were using vi to do "real work." It also meant that the accounting information did not track the usage of the program correctly. The moral is, you cannot assume that the name of the command reflects its function.

The pacct file format has remained largely unchanged since the early days of UNIX. The times that are CPU ticks could be immense but are passed over in a 16-bit word as a floating-point number, with a 3-bit octal exponent and 13 bits of fractional information.

There are two sets of programs available on your systems to process the pacct and wtmp data. The older program suite consisted of two programs: ac, which generated connect time accounts from the wtmp file; and sa, which analyzed the pacct file giving per-user and per-command statistics. These two commands were a little rudimentary if you wanted to maintain historical data. I certainly remember us having to write our own package because of these deficiencies, so we could prove who was using the machine.

The alternative to sa and ac is the standard System V accounting package, available on both Solaris and SunOS. Only SunOS supports the older commands. The System V package was written quite early on, perhaps in the early '80s. I think that it was up and running in System III. It's actually quite an impressive bit of work that consists of several programs and shell scripts that are intended to run unattended, recovering from errors without human intervention.

You can find all the programs and scripts on /usr/lib/acct. The central part of the package is the runacct script that performs daily processing, compressing data into (mostly text) summary files that are stored on /usr/adm/acct. Each month, the monacct script is run, and this takes the daily reports and further compresses them to create a set of monthly reports that contain summaries of terminal usage, command usage, disk usage and the time of last login. The format of the reports is self-explanatory.

If you are a student of shell scripts, the runacct command is interesting because it knows that it has several distinct sequential one-way tasks to perform and recognizes that it is possible for the machine to die while one of these states is being executed. It wants to be restartable, but restartable in the current state. To allow this reentrancy, it creates a state file holding a string that names the state and uses this to trigger the appropriate processing that is required.

The whole accounting package is an interesting example of how to construct complex tailorable systems from scripts and small programs. I am dubious that the package deals with the /usr/adm/wtmpx and /usr/adm/utmpx files correctly. I've had no time to pursue this.

Further Information

Thanks to Mike Barrow who mailed me from Spain with some suggestions for article topics. I'm not sure this was exactly what you had in mind, Mike.

The accounting system is well documented in Sun's System Administration Answerbook CDs. If you want a printed resource, then I suggest you read *Unix System Administration Handbook* by Nemeth, Snyder, Seebass and Hein (now in its second edition), published by Prentice Hall, ISBN 0-13-151051-7.

Another good book is *Essential System Administration* by Æleen Frisch, published by O'Reilly and Associates Inc., ISBN 1-56592-127-5. This book has a good diagram of the System V accounting system, showing how all the parts fit together and how they are used.

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 a SPARCstation 2. Email:* pc@cpg.com.