COBB

October 1996 • $11.50
VOL. 2 NO. 10

INSIDE SOLARIS
Tips & techniques for users of SunSoft Solaris™

in this issue

Configuring your computer to page you

By Al Alexander

Imagine that you're at home one night, getting ready to go on a two-week vacation to a terrific beach. As you're packing your swimsuit and suntan lotion, your pager suddenly starts beeping. "Oh no," you think, "there goes my vacation."

You look at the pager, fearing the worst. "Who's calling me, and what do they want?" Then you see the number and realize "Aha! My computer just paged me, and it's telling me that the root partition is nearly full." To solve the problem, you quickly dial up your computer and move a directory from the root file system to one that has some extra space. You also send yourself some E-mail to remind yourself to reorganize some files when you return from your vacation.

You then continue packing and head to the sun, sand, and water. Your vacation was rescued because your computer had the intelligence to recognize the potential problem and to call you before a larger disaster occurred. In this article, we're going to show you how to turn this fantasy into a reality. All you need is a way to make your computer system(s) page you when it detects an impending disaster.

How did this happen?

A few years ago, I created a UNIX daemon named `monitor`. Its purpose is to monitor my computer systems and alert me if something bad is happening or is about to happen. If so, the computer should try to contact me. Early versions of `monitor` sent E-mail, displayed messages on the console, and sent printouts to a printer near my cubicle to alert me of potential problems.

Lately, as my computer system responsibilities have spread throughout multiple states (and countries), I extended `monitor` to fax me and to send messages to my pager.

This proactive approach to system management is much better than a reactionary, defensive approach. Rather than spending hours fixing problems that have already occurred, you can spend minutes to avert them.

This solution is actually very simple to implement. First, you need a process that monitors your system for impending doom, and you need a way for your computer to send you a page. For the basics on a system-monitoring daemon, see the article "Writing a Script to Monitor Your System" on page 9.

Solving the paging problem

In order to use this method of making your computer page you, three things are necessary:

1. A digital pager
2. A Hayes-compatible modem connected to your computer
3. The UUCP programs installed

INSIDE SOLARIS
Tips & techniques for users of SunSoft Solaris™

Inside Solaris (ISSN 1081-3314) is published monthly by The Cobb Group.

Prices
U.S. ..... $115/yr ($11.50 each)
Outside U.S. ..... $135/yr ($16.95 each)

Phone and Fax
US toll free ..... (800) 223-8720
UK toll free ..... (0800) 961897
Local ..... (502) 493-3300
Customer Relations fax ..... (502) 491-8050
Editorial Department fax ..... (502) 491-4200
Editor-in-Chief ..... (502) 493-3204

Address
Send your tips, special requests, and other correspondence to:

The Editor, *Inside Solaris*
9420 Bunsen Parkway, Suite 300
Louisville, KY 40220
Internet: inside_solaris@merlin.cobb.zd.com.

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

Customer Relations
9420 Bunsen Parkway, Suite 300
Louisville, KY 40220
Internet: cr@merlin.cobb.zd.com.

Staff
Editor-in-Chief ..... Marco C. Mason
Contributing Editor ..... Al Alexander
Production Artist ..... Liz Palmer
Editors ..... Linda Recktenwald
Karen S. Shields
Circulation Manager ..... Mike Schroeder
Editorial Director ..... Linda Baughman
VP/Publisher ..... Lou Armstrong
President ..... John A. Jenkins

Back Issues
To order back issues, call Customer Relations at (800) 223-8720. Back issues cost $11.50 each, $16.95 outside the US. We accept MasterCard, Visa, or American Express, or we can bill you.

Advertising
For information about advertising in Cobb Group journals, contact Tracee Bell Troutt at (800) 223-8720, ext. 430.

Postmaster
Second class postage paid in Louisville, KY.
Postmaster: Send address changes to:

Inside Solaris
P.O. Box 35160
Louisville, KY 40232

The system is very simple. The `monitor` program, not listed here, must detect system problems. When a problem occurs, the `monitor` program uses a UUCP program to call your pager and leave you a message. In order to keep things simple, we encode the message into a seven-digit number. The top four digits identify the system that has the problem, and the lower three digits specify the problem.

Sending a page

Once you've set up your system properly, you can initiate the page in several ways. The simplest is to use the `cu` command, like this:

```
$ cu SystemName
Connect failed: CALLER SCRIPT FAILED
```

Please note the error message that `cu` prints after the page is done. (The error message doesn't occur for about a minute after you enter the command.) You can safely ignore this error because it's a by-product of the way this method works.

A more complex way to issue the page is to use the `Uutry` command. Because the `Uutry` program isn't likely to be in your path, you need to specify its full pathname, like this:

```
$ /usr/lib/uucp/Uutry -r SystemName
/usr/lib/uucp/uucico -r1
➥-sSystemName  -f -x5 >/tmp/pgr0001
2>&1&
tmp=/tmp/pgr0001
name (SystemName) not found; return
➥FAIL
name (DEFAULT) not found; return
➥FAIL
name (OTHER) not found; return FAIL
attempting to open /var/uucp/.Admin/
➥account
Job grade to process -
conn(pgr0001)
Trying entry from '/etc/uucp/Systems'
➥- device type ACU.
...
```

Here, I've trimmed away much of the output. Please note that when you use `Uutry` to page your system, the `Uutry` command will wait for you to stop the program. Thus, it's not useful for automated paging.

This method of sending the page is interesting because you can see the dialog between Solaris and your modem. The `Uutry` program also shows you the best way to send a page: with the `uucico` program.

Here's why I believe the `uucico` program is the best method to use:

1. You can override any retry time limits with the `-r` option.
2. It doesn't emit an error message when it sends the page.
3. You can tell it not to fill the screen with diagnostic information.

Like `Uutry`, the `uucico` program probably isn't in your path. So to execute the `uucico` program, you should invoke it like this:

```
$ /usr/lib/uucp/uucico -r1
➥-sSystemName
```

Once you set up your system correctly, you'll be able to send pages easily. Now let's take a look at how to set up the system.

Setting up UUCP

In the previous examples, I used *SystemName* without any explanation. It turns out that this *SystemName* is the heart of the paging system. You simply create a set of fake systems in the */etc/uucp/Systems* file of each computer that can send a page. Each *SystemName* entry corresponds to a particular piece of problem code the system can send to you. Figure A shows some of the entries I placed in my */etc/uucp/Systems* file.

If you want a more detailed explanation of how to set up the */etc/uucp/Systems* file, you may want to read the article "Configuring the Devices and Systems Files for cu" in the May issue, as well as peruse the `man` pages for the UUCP programs.

For the purposes of this article, all you need to do is create lines starting with your fake system name, followed by "Any ACU Any", followed by the phone number to dial. We suggest you use pgr*WXYZ* as the

Figure A

```
pgr0001 Any ACU Any 5551212,,,,,0001001#
pgr0002 Any ACU Any 5551212,,,,,0001002#
pgr0003 Any ACU Any 5551212,,,,,0001003#
pgr0004 Any ACU Any 5551212,,,,,0001004#
pgr0005 Any ACU Any 5551212,,,,,0001005#
pgr0006 Any ACU Any 5551212,,,,,0001006#
pgr0007 Any ACU Any 5551212,,,,,0001007#
pgr0008 Any ACU Any 5551212,,,,,0001008#
pgr0009 Any ACU Any 5551212,,,,,0001009#
pgr0010 Any ACU Any 5551212,,,,,0001010#
```

This portion of an /etc/uucp/Systems *file assigns the phone number of your pager and the pager code to a fake system name.*

system name, where pgr is the fake pager system, and *WXYZ* is the problem code that the system is reporting.

The pager phone number

As you can see in Figure A, we're manipulating the phone number to do all the work. For the purposes of this article, let's assume that your digital pager number is 555-1212. The phone number for pgr0001 is 5551212,,,,,0001001#.

The first part of the number is straightforward—it simply dials 555-1212. For a Hayes-compatible phone, a comma tells the modem to pause for two seconds. Thus, the five commas tell the modem to pause for 10 seconds. Next, the modem emits the code 0001001. Finally, the modem emits a pound sign (#), just as if you'd typed it on your telephone keypad.

The code simply encodes the system number as four digits and the problem code as three digits. So this code tells us that computer system 1 had problem code 1.

One problem this paging system has is that it doesn't automatically hang up quickly. Luckily, the system connected at 555-1212 interprets the pound sign as a command to hang up.

You may have to tune the telephone number to work with your digital pager and phone system. For example, some systems require that you dial 9 to get an outside line. For this, you'll want to insert a 9, at the beginning of the phone number. Similarly, if you have call waiting, you'll want to insert the code that turns off call waiting for the next call. In my area, it's *70, so I'd add *70, at the start of the phone number. In both these cases, we add a comma after the prefix to allow the command to be processed before we emit the rest of the digits in the phone number. Some equipment needs this delay, or digits may be missed. For example, if we need to do both, we may define a system as

```
pgr0003 Any ACU Any 9,*70,5551212,,,,,0001003#
```

I used a 10-second delay because my pager system normally picks up on the second ring, then gives a short voice prompt. If you have a longer voice prompt, you may want to increase the timeout period.

Lacking elegance?

Two parts of this system aren't elegant. The first involves calling my pager number, waiting 10 seconds, then blindly sending the numeric code that indicates the system name and error code. In a more perfect world, I would wait for a voice response from the pager system I'm calling, then send the numeric code. However, using a simple UUCP dialing system, this isn't possible. Does that mean this solution isn't elegant?

I've tested my pager system very heavily and determined that, 99 times out of 100, the pager system is ready to receive numeric input in five seconds after I dial the last digit of the pager phone number. At 25 seconds after the last number has been dialed, if my pager hasn't received a numeric code, it hangs up. Given that window between five and 25 seconds, I've elected to wait 10 seconds before transmitting the computer/error code. It may not seem elegant, but it hasn't failed yet.

The second ugly part of the system is that I can't hang up the phone properly. After the computer makes the call and transmits the computer/error code sequence, the # signal tells the pager system to hang up on my calling computer, which it does. That works fine, but my calling computer never hangs up the phone on its end of the line. Again, this appears to be a problem, but in reality the phone company terminates the connection after a minute of inactivity. This causes the modem to tell UUCP that the connection died, so the problem takes care of itself. The modem is always ready to receive my call well before the time I can dial into the system.

Other considerations

If you decide to use this method heavily, you may fill your */etc/uucp/Systems* file with many fake systems that perform pages. Not only does this congest the */etc/uucp/Systems* file, but you'll have so many different codes, you won't be able to remember what they all mean.

An alternative is to have a single fake system in your */etc/uucp/Systems* file. Then, instead of having your script file page to one of many fake systems, you can always page to the same one. Then your script can send you E-mail describing which problem(s) have occurred. When you log in, you can just read your E-mail to determine the problem. For more information on using mail in a script, see the article "Make a Shell Script Mail You a Summary" in the July issue.

Conclusion

You can use this fairly simple method to have your computer system page you in the event of system emergencies. You can then dial in and avert a disaster. You can easily extend this approach to have your system page you for other events, including the completion of long batch jobs, intruder detection, etc. The rest is up to you! ❖

Alvin J. Alexander is an independent consultant specializing in UNIX and the Internet. He has worked on UNIX networks to support the Space Shuttle, international clients, and various Internet service providers. He has provided UNIX and Internet training to over 400 clients in the last three years.

SECURITY ALERT

CERT/AUSCERT advisory issued against admintool

By Marco C. Mason

If you're administering a system on which you're concerned about security, you may want to avoid using `admintool`. The Australian Computer Emergency Response Team (AUSCERT) has uncovered a possible security loophole in the `admintool` program. This advisory (AL-96.03) was also adopted by CERT as document CA-96.16.

What causes the problem?

The problem is that `admintool` doesn't handle some temporary files in a secure fashion. The `admintool` program uses these files to enforce locking in order to prevent the system files from being manipulated by two users at once. Because the temporary files aren't handled securely, it's possible to coerce `admintool` to write any file with the user ID of the process that writes the file.

What makes this problem severe is that with Solaris 2.5, `admintool` normally has the setUID permissions bit set to run as root. Thus, a clever hacker may be able to gain root access by running `admintool` in order to access its lock files. While previous versions of Solaris don't install `admintool` with the setUID permission, the hacker need only wait for the root user to access `admintool` to gain root privileges.

Locking out admintool

CERT advises that you not allow any users to run `admintool` until patches are available. You can do so by removing the execute privileges of `admintool` like this:

```
# chmod 400 /usr/bin/admintool
# ls -l /usr/bin/admintool
-r--------   1 root   sys   303516 Oct 27   1995
➥/usr/bin/admintool
```

You can read the full text of the CERT advisory by retrieving document *ftp://info.cert.org/pub/cert_advisories/CA-96.16.Solaris_admintool_vul* from the Internet. As this situation unfolds, you'll find updated information at *ftp://info.cert.org/pub/cert_advisories/CA-96.16.README*.

Conclusion

Other interesting information, such as advisories, security information, and contact information, is available at *http://www.cert.org/*, at *ftp://info.cert.org/pub/* and in the comp.security.announce newsgroup. You can also join the CERT mailing list for other late-breaking news by sending your E-mail address to *cert-advisory-request@cert.org*. ❖

Scheduling a job for periodic execution

The whole point of having a computer is having it automate repetitive tasks. As system administrators, you should always strive to make your job simpler by automating everything possible. Scripts are great tools because they free you from the keyboard. Rather than typing dozens of commands and possibly making typing errors, you simply type the name of the script, and the system goes about its business.

You could also arrange it so that your computer decides when to run the program for you so you don't have to start the script manually. That way, you wouldn't even have to come to the office to run your month-end reports. In this article, we'll show you how to use `cron` to schedule jobs on a periodic basis.

The cron daemon

When you start Solaris, it starts a process named `cron` that manages scheduled jobs. Periodically, `cron` examines the files in */var/spool/cron/crontabs* to find out which, if any, jobs need to be run. If it's time to run a job, `cron` executes it.

In order to schedule the jobs, you need to create a file that describes what to do and when to do it. Figure A illustrates the format of a job request for `cron`. You may have as many job requests as you want.

If you don't want to specify any particular value for a field, put an asterisk in it. This tells `cron` to match it with all legal values. Thus, if you put an asterisk in each field, except the first in which you put a 5, you'll have a command that executes every time it's five minutes past the hour.

You can also use ranges and lists of values for each field. To specify a range, simply use two values separated by a hyphen. Values separated by commas indicate individual values. This allows you to build quite complex schedules.

Suppose you want to execute a command every 10 minutes during the workday, excluding the lunch hour. Obviously, you would also want to exclude Saturday and Sunday. You could do so like this:

```
0,10,20,30,40,50 8-11,13-17 * * 1-5 command
```

Here, we specified 0,10,20,30,40,50 for the minutes field, telling `cron` to execute the program every 10 minutes. The hours field is a bit trickier. Here we use two ranges, specified as: 8-11,13-17. This has the effect of excluding 12:00 through 12:59.

Next, we indicate that it may run on any day of any month. The next field limits `crontab` by the day of the week. Here, we specify Monday through Friday.

Sometimes, a command's desired schedule can be so complex you just can't write a `cron` entry to express it. So you break it down into multiple entries. There's no reason you can't run the same command with different entries.

For example, suppose you have a program you want to run every hour on the weekends, but only twice a day during the week. This is clearly impossible with a single `crontab` entry. However, these two `crontab` entries will do the trick nicely:

```
0 * * * 0,6 ComplexJob
0 0,12 * * 1-5 ComplexJob
```

The first entry takes care of executing `ComplexJob` hourly on Sunday (day 0) and Saturday (day 6). The second entry runs `ComplexJob` at noon and midnight from Monday to Friday.

The command field can hold any command you'd normally type at the shell prompt. When `cron` executes your job, it starts a `sh` job at your home directory and executes the specified statement. If you use a % symbol, `cron` will replace it with a new line. If you want a % symbol, precede it with a \.

Figure A

Column	Meaning	Range
1	minute	0-59
2	hour	0-23
3	day of month	1-31
4	month	1-12
5	day of week	0 (Sunday) - 6
6	command	any valid command

Each record allows you to specify when to execute any particular list of commands.

Changing your cron jobs

The `cron` daemon looks in the */var/spool/cron/crontabs* directory for each user's scheduled

jobs. One file exists for each user who has one or more scheduled jobs. The name of each file is the username that requested the scheduled jobs. Because users may have confidential information in their scheduled job requests, you can't simply modify the files in */var/spool/cron/crontabs*.

You can put a batch of scheduled jobs in this directory with the `crontab` command. To do so, you simply build your list of job requests and execute `crontab`, which will place your requested job batch in the */var/spool/cron/crontabs* directory.

Please note that your requested job batch replaces any job batch that you may already have. So if you want to keep any jobs in a previous batch, you must be sure to include them in the current batch. The easiest way to ensure that you don't accidentally delete important jobs is to ask `crontab` for a copy of the current job batch. You can then add any new job requests to the end of this file. Then when you tell `crontab` to submit the batch of job requests, you can be sure that you didn't omit any. You can display the list of job requests like this:

```
$ crontab -l
```

Miscellaneous notes

Keep in mind that when `cron` executes your jobs, it isn't logging in as you. Therefore, you must be careful what assumptions you're making, such as the default path. To ensure that your `cron` jobs succeed, you should probably write scripts to do the jobs. Be sure to explicitly set any environment variables, such as the path, at the beginning of your scripts.

If your `cron` job emits any information on the standard output or error streams, it will be packaged and E-mailed to you. Until you've completed debugging your `cron` jobs, this is a very useful tool. However, once you've debugged your `cron` job, you probably don't want this E-mail. To stop it, you must ensure that your `cron` job emits no information on these streams. The simplest way is to redirect these streams to the */dev/null* device.

One way to simplify things is to use a shell script for all but the most trivial `cron` jobs. If you use a script, you'll find it easier to test your `cron` jobs, because you can set all the environment variables required at the start. You can then test your script more simply. In addition, it's simpler to redirect the standard error and output streams on your `cron` line. Once you get your script working correctly, test it with a stripped-down account to ensure that you haven't depended on something that you've customized in your environment.

If you want, you can modify some of `cron`'s default settings by editing the */etc/default/cron* file. Here, you can specify the default path for root jobs and all others. You specify the default path for root jobs with `SUPATH`, and all other jobs get the path specified by `PATH`. You can also tell `cron` whether to keep a log of its activities by setting the value of `CRONLOG`. Suppose for a moment that you don't want `cron` to keep a log, and you want to run jobs with the path of */usr/bin:/usr/bin/local*. You can do so by setting the */etc/default/cron* file to this:

```
CRONLOG=yes
PATH=/usr/bin:/usr/bin/local
```

Please note that changing the path can be a security hazard. The most obvious security problem arises when you put a directory in the path into which an aspiring hacker can place files. Then it's a simple matter of writing a subversive script with the name of a file that's likely to be used in a `cron` job.

For this reason, it's a better idea to leave the default path alone and specify it only for the `cron` jobs that require it. Even then, be sure that the path you use doesn't have any potential for headaches. The normal path for user `cron` jobs is */usr/bin*, while root `cron` jobs use a path of */usr/sbin:/usr/bin*.

An example

As an example of how to use `cron` and the `crontab` command, let's submit a couple of `cron` jobs. The first job will be to send an E-mail message to remind us to pay our bills on the first of the month. The next job will be to execute the `ISOL_Monitor` script every 10 minutes. (We present this script in the article "Writing a Script to Monitor Your System" on page 9.)

First, you need to use `crontab` to retrieve the current list of `cron` jobs you have, like so:

```
$ crontab -l >MyCron
```

(At this point, your `crontab` file is probably empty, but it's better to be safe than sorry.) Then, invoke your favorite editor and add these two lines to the end of the *MyCron* file.

```
0 6 1 * * echo "Pay your bills!"
0,10,20,30,40,50 * * * * ISOL_Monitor >/dev/
  ➥null
```

Finally, to submit these `cron` jobs, type

```
$ crontab <MyCron
```

As you can see, at 6:00 a.m. on the first of each month, the first job simply echoes the message "Pay your bills!" Since it's creating output, `cron` will automatically package this output and E-mail it to us. The second job tells `cron` to execute the `cron` script every 10 minutes. The `cron` script generates a report each time it's run. Since we don't want to wade through 144 E-mail messages a day, we pipe the output to */dev/null* so that we aren't overwhelmed by the reports. (The `ISOL_Monitor` script will automatically page us when a problem occurs and E-mail us with the problem anyway.)

Conclusion

You shouldn't be a slave to your computer. Remembering to run a script every day at noon is a hassle, but one that your computer can easily handle. Rather than tying yourself down to a clock and your computer, use `cron` to make your computer watch the clock for you. ❖

EXPLORING LOG FILES

Tracking user logins

By Marco C. Mason

Last month, we talked about trimming back log files, lest they become too long. This calls to mind the obvious question, "What do we do with log files?" In our examples, we showed you the */var/adm/wtmp* file, which tracks all the user logins on a system.

The */var/adm/wtmp* file tracks the accounting information for all users. This means it tracks the users' names, the ports they logged in on, the times they logged in, the times they logged out, etc. It also tracks other things, such as time changes in the system.

Reading */var/adm/wtmp*

However, */var/adm/wtmp* is a binary file. So you can't just use `more` or `awk` to examine the accounting information. In order for you to access this information, Sun provides the `fwtmp` command. Because this command is related to system accounting features, it's located in the */usr/lib/acct* directory.

You can use this command to convert the binary formatted file to ASCII and back again. Thus, you can view the */var/adm/wtmp* file by simply creating an ASCII version and using `more` to display it, like this:

```
$ /usr/lib/acct/fwtmp </var/adm/wtmp >wtmp.data
$ more wtmp.data
```

When you do so, you'll see that the *wtmp.data* file contains many columns of data, as shown in Figure A. Since this file has no column headers, it's nearly impossible to tell what some of the columns mean.

Figure A

```
                system boot              0    2 0000 0000 837220087 Fri Jul 12 21:08:07 1996
                run-level 3              0    1 0063 0123 837220087 Fri Jul 12 21:08:07 1996
rc2             s2                       62   5 0000 0000 837220087 Fri Jul 12 21:08:07 1996
rc2             s2                       62   8 0000 0000 837220123 Fri Jul 12 21:08:43 1996
rc3             s3                       208  5 0000 0000 837220123 Fri Jul 12 21:08:43 1996
rc3             s3                       208  8 0000 0000 837220123 Fri Jul 12 21:08:43 1996
sac             sc                       228  5 0000 0000 837220123 Fri Jul 12 21:08:43 1996
ttymon          co                       229  5 0000 0000 837220123 Fri Jul 12 21:08:43 1996
LOGIN           co      console          229  6 0000 0000 837220123 Fri Jul 12 21:08:43 1996
zsmon           PM10                     231  6 0000 0000 837220124 Fri Jul 12 21:08:44 1996
marco           co      console          229  7 0000 0000 837220185 Fri Jul 12 21:09:45 1996
marco           co      console          229  8 0000 0000 837222753 Fri Jul 12 21:52:33 1996
ttymon          co      console          300  5 0000 0000 837222753 Fri Jul 12 21:52:33 1996
LOGIN           co      console          300  6 0000 0000 837222753 Fri Jul 12 21:52:33 1996
root            co      console          300  7 0000 0000 837222756 Fri Jul 12 21:52:36 1996
sac             sc                       228  8 0017 0000 837222781 Fri Jul 12 21:53:01 1996
root            co      console          300  8 0011 0000 837222781 Fri Jul 12 21:53:01 1996
                run-level S              1    1 0123 0063 837222781 Fri Jul 12 21:53:01 1996
                run-level S              1    1 0123 0063 837222781 Fri Jul 12 21:53:01 1996
shutdown                ~                0    0 0000 0000 837222806 Fri Jul 12 21:53:26 1996
```

This is a sample of the output from the `fwtmp` command.

Fortunately, you can find the format of the binary data by executing the `man wtmp` command. The columns in the */var/adm/wtmp* record have a one-to-one correspondence with the fields in the binary record. For your convenience, Figure B shows the record layout.

Figure B

```
struct utmp
    {
    char  ut_user[8];          /* user login name */
    char  ut_id[4];            /* /sbin/inittab id (created by */
                               /* process that puts entry in utmp) */
    char  ut_line[12];         /* device name (console, lnxx) */
    short ut_pid;              /* process id */
    short ut_type;             /* type of entry */
    struct exit_status
        {
        short e_termination;   /* process termination status */
        short e_exit;          /* process exit status */
        } ut_exit;             /* exit status of a process
                               /* marked as DEAD_PROCESS */
    time_t ut_time;            /* time entry was made */
    };
```

This is the binary format used in the /var/adm/wtmp *file.*

In Figure A, the fifth column is the first one you need to understand. It tells us what the rest of the record means. Table A shows you the different record types Solaris uses. However, the ones you'll be most interested in are 7 and 8, which you can use to tell when a user logs in and out of the system.

For record types 7 and 8, the first column specifies the user login name. For other record types, the first column may specify a process name other than a user logging in or out of the system. In Figure A, for example, you'll see that while only the users root and marco logged in, other processes, such as rc2, rc3, and sac did work as well.

When `init` starts a process, it fills the second column with the ID of the */etc/inittab* record that describes the process. Some processes started by `init` may fill in this field when starting other processes. Normally, the third column holds the device name the user logged in on. For record types other than 7, the system may put messages here, such as "run-level" or "system boot." The fourth column holds the process ID number.

The sixth and seventh columns aren't that useful to us. The sixth column holds the process termination status, and the seventh holds the process exit status. The eighth column contains a large integer value, which represents the time of the event, measured in seconds from January 1, 1970. Since Solaris uses a long integer to store the time, it can store the number of seconds for either 68 or 136 years, depending on whether the value is signed or not. Thus, we won't have a problem before 2038. The remaining columns simply show the time in a more easily readable format.

Writing /var/adm/wtmp

The `fwtmp` program can not only convert the */var/adm/wtmp* file to ASCII, but it can convert it back. This capability can be useful if you need to hand-edit the files. Please note that we don't recommend that you edit the files; we're just informing you that it's possible.

If you need to convert an ASCII data file back to binary, you can use the `fwtmp` program with the `-ic` switch. This tells `fwtmp` that the

Table A

Type	Description
0	Empty
1	Run Level—Tells the new run level when switching levels
2	Boot Time—The time at which the system boots
3	Old Time—The date and time from which you changed the system time
4	New Time—The date and time you set the system to
5	Init Process—The time `init` starts a process
6	Login Process—Processes used to control login
7	User Process—The time at which a user logs into the system
8	Dead Process—The time at which a process ends
9	Accounting—A record used by the accounting system

Of these record types in the /var/adm/wtmp file, the most interesting to us are 7 and 8.

input is going to be ASCII and the output will be binary. You can convert a file like this:

```
$ /usr/lib/acct/fwtmp -ic <wtmp.data
  ➥>/var/adm/wtmp
```

Keep in mind that if you convert the file back to binary, any records added to */var/adm/wtmp* since you originally converted it to ASCII will be lost. So if you're using system accounting, you'll lose information, unless you can find a quiet time on the system to perform any required edits.

Conclusion

With the information we've just presented, you can now keep track of people who use your system without learning how to use all the accounting scripts. If you've ever been curious about the */var/adm/wtmp* file, you should now have a good grasp on it. ❖

SYSTEM ADMINISTRATION

Writing a script to monitor your system

By Marco C. Mason

No matter what precautions you take, occasions always pop up when you need to drop whatever you're doing and attend to some urgent task. You just can't avoid it. As long as people use computers, computers will run out of resources.

As you know, once a computer runs out of resources, such as disk or swap space, recovery can be difficult. If the system crashes, you may spend hours getting it running correctly again.

If you could keep a close eye on your system, you could find out when a catastrophe is imminent and take steps to avert it. Any system administrator would gladly spend a few minutes to prevent a multihour recovery operation. In this article, we'll show you how to build some tools that help you monitor your system and prevent major breakdowns.

What do you want to monitor?

The resources you should monitor vary depending on the applications you run and the physical parameters of the system. Some of the situations you may want a warning about include:

- Low disk space
- Low swap space
- Too many processes
- Persistent warning messages
- Very high CPU usage over extended periods
- A high rate of network errors
- Heavy disk usage

The monitoring script we create in this article will monitor disk space on the / and */tmp* file systems, as well as available swap space and the number of processes running. Using the basic template we put together here, you can monitor as many things as you like.

Monitoring free space on a file system

One common cause of program failure is a file system running out of space. The amount of free space on a file system normally decreases as you accumulate data. You can use the `df` command to see how much disk space is free on all your file systems, like this:

```
# df -b /
Filesystem             avail
/dev/dsk/c0t0d0s0      354283
```

As you can see, the / file system has about 350 megabytes of free space. You can examine the output of the `df` command to decide which file systems are too full for comfort.

For the purposes of the script that we'll put together later, we want only the number in the second column of the second line. To do so, we pipe the results of `df -b` to `awk`, telling it that we want only the second field on the second line, like this:

```
$ df -b / | awk 'NR==2 { print $2 }'
354283
```

We can place this value in a variable by enclosing the expression in grave accents (`)

and treating it just like a value in an assignment statement. The completed statement that puts the free space of the / directory into the TEMP variable is

```
TEMP=`df -b / | awk 'NR==2 { print $2 }' `
```

Monitoring free swap space

Another major catastrophe occurs when the system runs out of swap space. In this case, Solaris must start killing jobs to free swap space. Since Solaris doesn't know which jobs are the most important, it may easily kill your mission-critical jobs.

If you've installed Solaris in the normal way, the swap area shares a disk slice with the */tmp* file system. In this case, you don't necessarily need any special code to check for low swap space. Instead, you can simply use the check for low free space on the */tmp* file system.

On the other hand, if you've separated the swap space from the */tmp* file system, you need a different method of finding out how much free swap space you have. In this case, you can use the `swap -l` command to list the swap areas, like this:

```
$ swap -l
swapfile             dev    swaplo  blocks  free
/dev/dsk/c0t0d0s1    102,1  8       131752  110784
/extra_swap          -      8       992     992
/extra_swap_2        -      8       1408    1408
```

As you can see here, this system has three swap areas, with a total of 113,184 blocks free (nearly 60MB). If you're going to write a script to monitor your swap space, all you need to do is add the amount of free space for all swapping partitions and compare the result to a threshold value to see if you're running dangerously low.

You can pipe the output of `swap -l` to a simple `awk` script to compute the total free space. The `awk` script must simply add together all the values in the fifth column for all lines after the first. At the command line, type the following command to get the amount of free swap space:

```
$ swap -l | awk 'BEGIN {ttl=0} NR>1 {ttl+=$5}
➥END {print ttl}'
113184
```

As you'd expect, we can place the amount of free swap space in a shell variable by enclosing the preceding expression in grave quotes and making the assignment, like this:

```
TEMP=`swap -l | awk 'BEGIN {ttl=0}
➥NR>1 {ttl+=$5} END {print ttl}'`
```

How many processes are running?

Perhaps your system has a problem when too many processes are executing at once. If so, you may want to monitor the number of processes executing at any given time. Counting the number of active processes on the system is easy. We use the `ps -A` command to report all processes, one per line. Then we use `wc -l` to count the number of lines, as follows:

```
# ps -A | wc -l
      44
```

So, to put the number of processes in a shell variable, we can use this command:

```
TEMP=`ps -A | wc -l`
```

Checking your system state with a shell script

You can check for many other things, but this is a good start for our system-monitoring script. Once we've obtained the information we want, we use basically the same structure to determine whether the system is in trouble. We use an `if` statement to see whether we've violated the limit. If we have, we append a warning message to a report file and set the `STATUS` variable to 1, as shown in Listing A.

Listing A

```
if [ ${TEMP} -lt MIN_ROOT_SPC ]; then
   echo "   Not enough!"
   cat <<- XYZZY >>${REPORT}
        Insufficient space on /
          (${TEMP} < ${MIN_ROOT_SPC})

        XYZZY
   STATUS=1
fi
```

We use this ISOL_Monitor structure throughout to warn the user about potential problems.

The blue lines of code use a *here document*, as described in the article "Automating Applications that Accept User Input" in the June issue. These lines add a failure warning record to the file specified by `REPORT`.

Finally, after the script checks all parameters, it decides whether to send E-mail and page the system administrator. It then deletes the temporary file it used to build the mail message. (On an early version of the *ISOL_Monitor* script, we inadvertently tested it. We forgot to delete the temporary file, and eventually the script told us that the */tmp* file system was too full!)

```
if [ ${STATUS} -gt 0 ]; then
   mail ${SYSADMIN} <${REPORT}
   cu pgr_${SYSADMIN} >/dev/null
fi
rm ${REPORT}
```

Please note that for our purposes, we're assuming you created a paging system named pgr_*SysAdmin*, where *SysAdmin* is the username of your system administrator.

Listing B

```
#! /usr/bin/ksh
#-----------------------------------
# Monitor system statistics, and warn
# sysadmin(s) of any impending probs.
#-----------------------------------

# CONFIGURATION
MIN_ROOT_SPC=1000000
MIN_TEMP_SPC=2000000
MIN_SWAP_SPC=1000000
MAX_PROCS=3
SYSADMIN=marco
PATH=/usr/sbin:/usr/bin

# By default, we're not going to send a
# page, or any E-Mail
STATUS=0
REPORT=/tmp/ISOL_Monitor_${$}
rm ${REPORT}

# Is there enough space on /?
TEMP=`df -b / | awk 'NR==2 { print $2 }' `
echo ${TEMP} "blocks left on /"
if [ ${TEMP} -lt MIN_ROOT_SPC ]; then
   echo "   Not enough!"
   cat <<- XYZZY >>${REPORT}
      Insufficient space on /
         (${TEMP} < ${MIN_ROOT_SPC})

      XYZZY
   STATUS=1
fi

# Is there enough space on /tmp?
TEMP=`df -b /tmp | awk 'NR==2 { print $2 }' `
echo ${TEMP} "blocks left on /tmp"
if [ ${TEMP} -lt MIN_TEMP_SPC ]; then
   echo "   Not enough!"
   cat <<- XYZZY >>${REPORT}
      Insufficient space on /tmp
         (${TEMP} < ${MIN_TEMP_SPC})
      XYZZY
   STATUS=1
fi

# Is there enough swap space?
TEMP=`swap -l | awk 'BEGIN { total=0 } NR>=2 { total += $5 }
➥END { print total }' `
echo ${TEMP} "blocks of swap space left"
if [ ${TEMP} -lt MIN_SWAP_SPC ]; then
   echo "   Not enough!"
   cat <<- XYZZY >>${REPORT}
      Insufficient swap space
         (${TEMP} < ${MIN_SWAP_SPC})

      XYZZY
   STATUS=1

fi

# Are there too many processes running?
TEMP=`ps -A | wc -l`
echo ${TEMP} "processes currently running"
if [ ${TEMP} -gt MAX_PROCS ]; then
   echo "   Too many!"
   cat <<- XYZZY >>${REPORT}
      Too many processes!
         (${TEMP} > ${MAX_PROCS})

      XYZZY
   STATUS=1
fi

# If we've detected any bad problems,
# E-Mail the report to the sysadmin
# and then issue a page
if [ ${STATUS} -gt 0 ]; then
   mail ${SYSADMIN} <${REPORT}
   cu pgr_${SYSADMIN} >/dev/null
fi
rm ${REPORT}
```

The ISOL_Monitor *script monitors your system and alerts you when a resource is critically low.*

Listing B on the previous page, shows the entire *ISOL_Monitor* script we created to monitor the system and evaluate the results. The configuration section at the beginning sets the limits we're going to complain about if violated.

As you can see, we set obviously bad limits in order that you might see the script send you E-mail and page you. Also note that you need to change the `SYSADMIN` variable to your username. Once you install the script on your system, just tune these parameters to values that suit your needs.

Miscellaneous

The article "Configuring Your Computer to Page You," on page 1, shows how to set up your computer so it can page you when the *ISOL_Monitor* script detects a problem. You will probably want to execute this script frequently. For information on setting the script to run automatically, check out the article "Scheduling a Job for Periodic Execution" found on page 5". Finally, we rely heavily on `awk` for making this script work, so you'll want to refer to the `man` page on `awk` or read the article "An Introduction to `awk`" in our May issue.)

Conclusion

The *ISOL_Monitor* script we presented here is only a starting point on which you can build a more sophisticated monitoring system. There are many ways you can improve it. Here are some suggestions:

1. Monitor the system for other potential problems, such as missing processes.
2. Use `cron` to schedule the script for frequent execution.
3. For sites with multiple shifts, set the `SYSADMIN` variable based on the time of day.
4. For sites with external access, monitor hacking attempts. ❖

Marco C. Mason is a freelance computer consultant and author based in Louisville, Kentucky.

HARDWARE CONFIGURATION

Enabling additional serial ports on Solaris x86

You've just read the article "Configuring Your Computer to Page You," on page 1 and you think it's a good idea. You even have an old Hayes 1200-baud modem in the closet you haven't used for years. "Hmmm," you think, "that would be cool!" So you dig your modem out of the closet, plug it into your Solaris x86 box, set up your */etc/uucp/Systems* file, and type

```
$ cu pgr0001
Connect failed: CAN'T ACCESS DEVICE
```

Hey! That's not the message you were expecting. Why isn't your system paging you? Usually, this message is telling you that Solaris can't access your serial port. When you first install Solaris x86, it tries to configure *tty00* as connected to COM1 so you can use the mouse in OpenWin and CDE. By default, any additional serial ports on your machine are disabled. Since you're (presumably) still using your mouse, you probably plugged your modem in another COM port. We'll show you how to configure Solaris x86 for additional serial ports.

The *asy.conf* file

In order to make Solaris recognize your hardware, you have to understand how the asynchronous port configuration file, named *asy.conf*, works. This file contains a record for each serial port you're trying to configure, and it describes to Solaris x86 the hardware you want it to recognize.

Please note: The driver *asy* assumes that your hardware is a standard PC-compatible serial port, i.e., one based on the 8250, 16450, or 16550 UART. If you're trying to add a different type of serial port, you'll need a software driver for it.

Now let's take a look at the format of the *asy.conf* file. Figure A shows a typical entry for a serial port. Note that it's a list of name=value pairs terminated with a semicolon.

Figure A

```
name="asy" class="sysbus" interrupts=12,4
➥reg=0x3f8,0,0 ioaddr=0x3f8;
```

This asy.conf entry describes a serial port using IRQ 4 with I/O ports starting at 0x3f8.

The first field, `name`, tells which driver to use for the serial port, `asy` in this case. *Don't change this!* The second field, `class`, tells the generic type of driver being used.

The `interrupts` field describes the interrupts used for this port. This is always a pair of numbers separated by a comma. You should *never* change the first number. This number tells Solaris what priority the interrupts are, between 1 and 16, where the lower numbers are the higher priorities.

If you're configuring a high-speed port, it may be tempting to increase the interrupt priority (i.e., decrease the first number) to ensure that you don't lose characters. However, *don't succumb to this temptation!* If you do, you may make your system unstable, since more important interrupts may not get serviced quickly enough.

The second number is the IRQ number used by the port, a number from 0 to 15. If you configure your machine like most standard PC-compatible computers, your serial ports are most likely set up as shown in Figure B.

Figure B

Port	IRQ	I/O address
COM1	4	0x3f8
COM2	3	0x2f8
COM3	4	0x3e8
COM4	3	0x2e8

Most PC-compatible computers set up their serial ports like this.

The `reg` parameter tells the `asy` driver the address for the UART controlling the serial port. The address is expressed as a set of three numbers separated by commas, where the first number, if non-zero, tells the driver the serial port is connected to a set of I/O ports. If this value is non-zero, you should use the I/O base address. The second and third numbers, if non-zero, tell `asy` the UART is memory mapped. PC-compatible serial ports are always mapped to I/O ports, so you'll always use zero for the second two numbers. Figure B shows the I/O addresses usually used for serial ports.

The final parameter, `ioaddr`, specifies the base I/O address for the serial port. This value specifies the lowest I/O port used by the serial port. Again, Figure B contains the information for the default PC-compatible configuration.

Special note on interrupt sharing

While Figure B shows that four ports use only two IRQs, you can't always count on it working that way. Normal PC hardware can let your serial ports share interrupts, but *only if they're on the same card*, and sometimes, not even then.

As an example, suppose your motherboard has two serial ports, and you have a serial card that provides two more serial ports. If you configure the motherboard ports as COM1 and COM2 and the serial card ports as COM3 and COM4 using the PC-compatible interrupts, you'll have a problem. In this case, a better solution would be to configure the motherboard ports as COM1 and COM3 and put COM2 and COM4 on the serial card.

However, even this may not work in some situations. In this case, you must use other IRQ channels for the ports that don't work. For this case, you'll have to explore alternate configurations.

Alternate configurations

What can you do if you need more than four ports? What if your hardware won't support IRQ sharing for your COM ports? And what if you have an I/O conflict with another device?

In all these cases, you have to customize your *asy.conf* file. To do so, you need to find a suitable IRQ and I/O base address for each of your ports. As your first step, you need to find out which IRQ ports are already in use, so you can see which ones remain. Then you can decide which port will get which IRQ address.

Next, you need to map out which I/O addresses are available to you. Keep in mind that the UARTs found on a standard serial card require eight consecutive I/O addresses, and serial cards restrict the ranges you can choose.

Many serial cards are limited to their IRQ selection. You may have to buy a "high IRQ" serial card to access the IRQs you need. Once you decide on your IRQ and I/O base selection, you can modify the *asy.conf* file accordingly.

As an example, suppose you want to add a fifth serial port to your computer, and you've configured a COM port at I/O address 0x190 with IRQ 11. You would then write your *asy.conf* line as:

```
name="asy" class="sysbus" interrupts=12,11
➥reg=0x190,0,0 ioaddr=0x190;
```

Ordering in the *asy.conf* file

In our discussions of ports and IRQs, we called the ports by their traditional BIOS names, like COM1, rather than Solaris' device names, like *tty00*. Why did we do this?

It turns out that Solaris assigns `tty` numbers based on the order of entries in the *asy.conf* file. The first entry specifies *tty00*, the next specifies *tty01*, and so on. Thus, you can configure any `tty` port to be any COM port. This is one reason you must be careful when reconfiguring serial ports.

Normally, the *asy.conf* file comes with default definitions for the four standard PC-compatible COM ports, in order. If you start your system and configure it with COM1 and COM4, all will be fine. COM1 will be *tty00* and COM4 will be *tty01*. If you go back and add COM2 later, and you don't rearrange the entries, then COM2 becomes *tty01*, and COM4 becomes *tty02*!

You should try to leave the first COM port alone, if possible. Otherwise your mouse will stop working in OpenWin and CDE.

Where's the *asy.conf* file?

Before Solaris 2.5, the *asy.conf* file was located in your */kernel/drv* directory. Starting with Solaris 2.5, Sun rearranged the file hierarchy to allow simpler support of multiplatform networks. Sun created the */platform* directory to support other platforms. Beneath this they've added a subdirectory based on the processor type, such as *sparc* and *i86pc*. Beneath this, it's business as usual, so the *asy.conf* file for Solaris x86 v2.5 is located at */platform/i86pc/kernel/drv*.

Restart Solaris

Now all you need to do is restart Solaris, telling it to reconfigure itself. To do so, execute the following commands as `su`:

```
$ touch /reconfigure
$ shutdown
```

When the system comes up, it will automatically configure itself with the new serial port configuration.

Conclusion

Now you know the basics of setting up additional serial ports for Solaris x86. If nothing else, you now understand what's happening in Solaris when you uncomment the *asy.conf* line for a standard COM port. ❖

USING SHELLS

Creating your own shell commands with alias

Everyone has his or her own programming preferences. UNIX accommodates you by being incredibly flexible. You have many ways of customizing UNIX to do what you want, in the way you want to do it. Sometimes, you only need to select the appropriate command-line switches on a program that already exists. Other times, you might reformat the output from a command using `awk`. You can also combine many commands using shell scripts. You can even create your own commands with C or some other programming language.

In this article, we'll show you how to use the `alias` command. The `alias` command isn't as powerful as a shell script, but it's more flexible than using the command line.

How does the alias command work?

First, you should know that `alias` command support isn't built into the Bourne shell.

However, full support exists in the Korn and C shells. When you enter a command, these shells first scan the list of aliases to see if the first word in the command you entered is listed. If so, the shell replaces the first word with the aliased value. An alias lasts as long as the current session, so if you want an alias to be in effect every time you log into your system, you'll need to edit your login files accordingly.

The `alias` command allows you to define a shorthand name for a command or a sequence of commands. You use the `alias` command like this

```
$ alias name=value
```

where *name* is the shorthand name and *value* is what you want to replace it with.

Thus, you can use `alias` for tasks as simple as renaming a command. For example, many system administrators assist people moving from DOS to UNIX by creating a couple of aliases to make their environment more familiar. These two aliases allow the user to use the familiar `dir` and `del` commands, as shown here:

```
$ alias dir=ls
$ alias del=rm
```

In this case, the `dir` command acts like `dir /w` under DOS. In DOS, the `dir` command normally acts more like `ls -l`. Thus, if we want to make things a little nicer for the DOS users, we could instead alias `dir` as

```
$ alias dir='ls -l'
```

This immediately suggests another use for the `alias` command. If you often execute a command with a particular set of arguments and/or options, you can simply create an alias for it and save quite a bit of typing.

Advanced aliasing

As we mentioned earlier, before the C or Korn shell executes a command line that you enter, it scans the alias list for a match of the first word and, if found, replaces the word with the alias value. Since you can execute multiple commands on a single command line by separating them with a semicolon, you can create an alias that executes a series of commands.

Suppose that you always like to see the amount of free space in a directory after the directory listing. You can create an alias that does so like this:

```
$ alias lsd='ls;pwd;df .'
```

This `lsd` alias first lists all the files in the current directory, then it displays the current directory, followed by the disk usage statistics on the current file system.

```
$ lsd
Hello  work.c   temp   a.out
/export/home/marco/work
/export/home   (dev/dsk/c0t1d0s7 ):
➥450132 blocks 233114 free
```

Whenever you execute multiple commands in a single alias, you must be careful. If any command other than the last one displays data to the screen, then redirecting your new command may produce surprising results. For example, let's redirect your command to a file, like this:

```
$ lsd >xyzzy
Hello  work.c   temp   a.out
/export/home/marco/work
```

As you can see, the first two commands displayed to the screen, and only the last command's output went to file *xyzzy*. Fortunately, there's a simple way around this. Just enclose the commands in a parenthesis, like this:

```
$ alias lsd='(ls;pwd;df .)'
```

This tells the shell to execute all the commands in a single subshell. Then the I/O redirection will affect the subshell rather than the individual commands.

There's another way to build a command from other commands. Using I/O redirection, you can build a pipeline of commands, piping the output of one command to the input of the next to get a job done. For example, to find a process started by a particular user or a process executing a particular command, we often use the construct

```
$ ps -ef | grep word
```

where *word* is the user name or command name of interest. The only part that changes each time we use it is *word*. Thus, we can create an alias named `fproc` to help us find a particular process, like this:

```
$ alias fproc='ps -ef | grep'
```

So, to find a process started by user marco, we can now type

```
$ fproc marco
   root   676   623  0  11:12:51  pts/5  0:00
➥grep marco
```

Keep in mind that the alias is a simple substitution. If you use alias to execute multiple commands, you won't be able to change the switches on any but the last command. For example, if you create the `dir` command as

```
$ alias dir='ls -l | more'
```

you can list the contents of the directory and then page through the directory entries with `more`. If you want to view all the entries starting with a period (.), you'll need to change your alias or enter the command line with the new parameters. Simply typing `dir -a` won't work, because the `-a` option affects `more`, not `ls`.

Removing an alias

Sometimes, you want to keep an alias only for the duration of a project. You don't want to keep hundreds of aliases around. So UNIX provides the `unalias` command to allow you to remove an alias from your account. To use it, you simply type

```
$ unalias name
```

where *name* is the alias you want to remove.

What aliases are defined?

You might want to know which aliases you've defined. You can find out by simply entering the alias command with no arguments. This tells `alias` to print a list of all the aliases currently defined in the system.

Security implications

Now that you know how the `alias` and `unalias` commands work, you'll probably want to start using them. However, you should be aware of the security implications of `alias`. If you leave your workstation unprotected, someone could create an alias on your account that could subvert system security.

Let's take a closer look. It's hard to imagine a day where you don't execute the `ls` command in the course of system administration. Suppose a nefarious user wrote a simple script named *GetRoot* that would create a root-level account. That user may be waiting for you to leave your seat to execute the command

```
$ alias ls='GetRoot&;ls'
```

Then the user simply clears your screen and waits for you to return. Chances are that you'll execute the `ls` command before you leave for the day. So our nefarious user will probably soon have a new root-level account and can then build some backdoors into your system.

Conclusion

You can use the `alias` command to make things simpler for you and the people who use your system. Not only can you customize UNIX commands that already exist, but you can put together simpler commands to build commands that you find more useful. You'll probably want to read the `man` pages for `ksh`, `csh`, and `alias` to find more information about using `alias`. ❖