67

AIX

May 2001

update

In this issue

AIX Update

Published by

Xephon
27-35 London Road
Newbury
Berkshire RG14 1JL
England
Telephone: 01635 38342
From USA: 01144 1635 38342
E-mail: trevore@xephon.com

North American office

Xephon
PO Box 350100
Westminster, CO 80035-0100
USA
Telephone: 303 410 9344

Subscriptions and back-issues

A year's subscription to *AIX Update*, comprising twelve monthly issues, costs £180.00 in the UK; $275.00 in the USA and Canada; £186.00 in Europe; £192.00 in Australasia and Japan; and £190.50 elsewhere. In all cases the price includes postage. Individual issues, starting with the November 1995 issue, are available separately to subscribers for £16.00 ($23.00) each including postage.

AIX Update on-line

Code from *AIX Update*, and complete issues in Acrobat PDF format, can be downloaded from our Web site at http://www.xephon.com/aixupdate.html; you will need to supply a word from the printed issue.

Editors

Trevor Eddolls and Richard Watson

Disclaimer

Readers are cautioned that, although the information in this journal is presented in good faith, neither Xephon nor the organizations or individuals that supplied information in this journal give any warranty or make any representations as to the accuracy of the material it contains. Neither Xephon nor the contributing organizations or individuals accept any liability of any kind howsoever arising out of the use of such material. Readers should satisfy themselves as to the correctness and relevance to their circumstances of all advice, information, code, JCL, scripts, and other contents of this journal before making any use of it.

Contributions

When Xephon is given copyright, articles published in *AIX Update* are paid for at the rate of £170 ($260) per 1000 words and £100 ($160) per 100 lines of code for the first 200 lines of original material. The remaining code is paid for at the rate of £50 ($80) per 100 lines. In addition, there is a flat fee of £30 ($50) per article. To find out more about contributing an article, without any obligation, please contact us at any of the addresses above and we will send you a copy of our *Notes for Contributors*, or you can download a copy from www.xephon.com/contnote.html.

Printed in England.

Bottleneck basics

It's a phone call most administrators never want to receive: the server is slow, no one can check e-mail, and Web pages are loading slowly or not at all! Too often administrators find themselves trying to climb up the steep slope of increased demand. As a user base grows, the demand placed on the server grows as well. This growth may be linear and predictable, or it may be completely random or exponential.

There are ways to avoid the angry phone call altogether. Understanding system bottlenecks and gathering statistical data can help you project your system's current and future needs. This can eliminate user complaints – and prevent that phone from ringing.

WHAT CAUSES A BOTTLENECK?

Why does a system slow down in the first place? Slowdowns can usually be attributed to one or more bottlenecks, which are caused when part of the system is not running fast enough to keep up with the demands placed on it. The most common bottlenecks occur for the following reasons:

- Slow disks or disk arrays aren't able to handle I/O requests quickly enough.
- The system is starved of memory, so applications are forced to swap to disk, which can slow response time drastically.
- The system is out of processor power.
- The network interface is overloaded.

So how can you tell which of these may be having a problem? By using the various tools of the capacity planning trade – **sar**, **netstat**, and **top**.

SAR

sar is by far one of the most valuable tools an administrator has to track past trends and predict future demand.

First you'll need to configure it to begin collecting data. Edit the system's **adm** crontab: **crontab -e adm**.

Remove the comments so that you have these lines:

```
0 8-17 * * 1-5 /usr/lib/sa/sa1 1200 3 &
0 * * * 0,6 /usr/lib/sa/sa1 &
0 18-7 * * 1-5 /usr/lib/sa/sa1
5 18 * * 1-5 /usr/lib/sa/sa2 -s 8:00 -e 18:01 -i 3600 -ubcwyaqvm &
```

This will enable **sar** for system activity reporting. Your system will now begin gathering data. For a detailed explanation of how to use **sar**, please see the **sar** information pages. Here's a quick list of **sar**'s more useful features:

- **sar** running with no options shows CPU usage.
- **sar -q** shows your average queue size.
- **sar -p** and **sar -g** show paging activity.
- **sar -d** shows disk utilization.
- **sar -f <filename>** reads a previously saved file.

NETSTAT

One of **sar**'s shortcomings is that it will not trend network traffic for you. This can be done using **netstat**. **netstat -in** will show you your network interfaces, how much traffic they have passed since booting, and any problems with them. For example:

```
netstat -in
Name Mtu Net/Dest Address Ipkts Ierrs Opkts Oerrs Collis Queue
en0 1500 192.168.100.0 192.168.100.1 1477758588 0 2897473608 0 0 0
en1 1500 192.168.101.0 192.168.101.1 3228181693 157415 3365694030 0 0 0
```

From this example, you can see that *en0* and *en1* are very busy, with *en1* having seen some incoming errors on its interface.

TOP

top was introduced in *AIX Update*, Issue 55, May 2000. It's a Unix utility that generates continuous reports on the state of your system,

including a list of the fifteen processes that are consuming the most CPU time.

SO WHERE'S THE SLOWDOWN?

Using tools such as **sar**, **netstat**, and **top** can help you determine where a slowdown might be happening, or where one is about to happen. Here are some examples of how you can use these tools:

- **sar** with no options

  This will show how idle the CPUs are. If your CPUs are using a lot of *%usr* or *%sys*, you may have to add extra CPUs to deal with increased demand. If *%wio* is high, your system is waiting for your I/O subsystems to catch up. You may have a slow disk or array.

- **sar -g**

  If you have many *pgscan*s, your system is swapping. *No* swapping is the only good swapping. Your system is probably short on memory. Use **sar -r** to verify this.

- **netstat -in**

  Look to see if an interface is overloaded with traffic. If so, you may have to add another physical interface. Also, look for *Ierrs*, *Oerrs*, and *Collis*. These should all be relatively low numbers if not zero. High numbers in these columns can indicate network problems such as speed or duplex autonegotiation issues, bad cabling, or a bad switch port.

- **top**

  If all else fails, look at **top**. What process is taking up the most resources?

ANALYSE THE DATA AND MAKE RECOMMENDATIONS

So you've put together all of your reporting tools. You're able to do past trend analysis and future growth predictions based on **sar**. You can also do real-time snapshots using **top**. What should you do to

make the system perform better now, as well as in the future?

It's very important to note that, if you do identify and solve a bottleneck, your solution can potentially cause even worse problems. For example, if you have idle CPU and a busy disk, replacing the busy disk with a fast disk can cause the CPU usage to spike. Remember, capacity planning is a constant exercise, not a one-time activity. Here are some scenarios:

- Busy I/O subsystems

  Say you've determined by using **sar -d** that one or more of your disks is very busy (more than 90 percent busy). Either move I/O from that disk to a faster disk or array, or split up the I/O amongst many arrays, depending on the data. Remember also that SCSI interfaces can be overloaded as well. This is difficult to determine, but it's a good idea to add new SCSI interfaces and balance I/O traffic accordingly. Improving I/O access can have a major impact on CPU or network performance.

- Busy CPUs

  Using **sar**, it may become apparent that your system is in heavy *%usr* and *%sys*. Adding CPUs in this situation can help, but it may not solve the problem. A poorly written application can consume infinite amounts of CPU resources.

- Busy network

  netstat -in may show your network interface to be very busy. Add another physical interface, but beware of increased I/O and CPU demands. Is the system swapping? Add more memory. Do whatever you can to prevent the system from swapping. If possible, create swap areas on fast disks.

APPLICATION SLOWDOWNS

Sometimes system hardware isn't the problem at all. Remember that applications are what consume system resources, and poorly written applications can be very difficult to deal with. Here is some advice:

- Beware of single-threaded applications. While a single-threaded application is generally easier to develop, it's also more costly to run. Many applications developed in-house are single-threaded. The worst example is the single-threaded non-forking application. This is an application that's not only single-threaded, but also won't fork copies of itself to consume resources more efficiently. **top** will show only one instance of this daemon running. **ps -eLf** will show only one thread. This can be a very challenging application, because it may consume only a single CPU even if you add more CPUs. Single-threaded applications that fork copies of themselves are much easier to deal with, but are still not as efficient as a multi-threaded application.
- Learn as much as possible about the application you're dealing with. Talk to the vendors or the authors because they'll know what tricks and tips will work best. Often, entries need to be made in */etc/system* so that an application can work at peak capacity. **ndd** settings may also need to be tweaked based on your current needs. Consider all of these performance suggestions before adding new hardware.

PLANNING FOR FUTURE CAPACITY

Sometimes the best way to plan for the future is to look at your past performance data. Using **sar**, you can ascertain a trend in the resource consumption on your system. If your system CPU was 90 percent idle three months ago, and now it's 80 percent idle, it's not unreasonable to assume that in three months your system will only have 70 percent idle CPU. Some parts of your system may grow at exponential rates, such as I/O or network subsystems. That's why it's important to constantly gather data, so you can see where you've been and where you're going. You may also want to consider writing scripts that can monitor **sar** and alert you when certain thresholds are reached. If your I/O is 70 percent busy for more than a week, it's probably time to consider a replacement or an upgrade.

Communication within your own organization can help you meet future capacity as well. You need to know if your marketing department

is planning a big push to acquire more customers, or if a new accounting system is going into place next week. Growth is then predictable, because you can plan for increased access to your database or for exponential growth in your Web server's traffic. Knowing how your customers will be using your servers will help you provide better performance.

SCALING HORIZONTALLY AND VERTICALLY

For large-scale applications, it's extremely important to be able to scale your systems both horizontally and vertically. Horizontal scaling allows you to add many boxes to serve the same application, while vertical scaling allows you to break the application into pieces so that each one can be scaled horizontally. A system designed to be both horizontally and vertically scalable allows you to add servers as demand increases. This way, you avoid the pitfalls of trying to scale one big box, and can benefit from having many small boxes.

Here are some examples of horizontal and vertical scaling:

- Horizontal Web servers

  Multiple Web servers are set up serving identical content, using independent hardware on different networks. DNS round robin or load balancing can be used.

- Horizontal and vertical e-mail solutions

  Each component of the e-mail server (mx, SMTP, POP, Web mail) can be run on its own independent server. Multiple individual servers can be set up to balance the load. In this way, you can have four mx servers, two SMTP servers, two POP servers, and one Web mail server, or whatever configuration you need to meet demand.

- Horizontal and vertical Web servers

  Multiple Web servers can be set up – some that serve graphics, and others that serve just CGI scripts. Servers can be added as demand increases.

Using reporting tools such as **sar** makes it possible to identify trends on your system. Learning about the applications on your system and communicating with your organization can also help when planning future growth. Finally, designing a system that can scale both horizontally and vertically can help you stay one step ahead of the growth curve.

Werner Klauser
Klauser Informatik (Switzerland)

Verifying fileset-level integrity

Lawrence Livermore National Laboratory has one of the largest IBM SP complexes in the world, including the current world's largest supercomputer, ASCI White. As a system administrator in this environment, it is important for me to be able to verify the integrity of our installed software across our SP nodes, and in some cases, between multiple SP clusters. While IBM has provided a program called *lppdiff* as part of the PSSP software package, I have found *lppdiff* to be both inefficient on large clusters, and unwieldy in its output.

As an alternative, here is a Perl script called *lppstat*, which can be used to easily and concisely identify fileset-level discrepancies between two systems. While this utility was written for use on our large SP clusters, it can be used to effectively compare fileset levels on any two AIX systems.

lppstat analyses the output of the command **lslpp -Lcq** from two systems. The syntax of the command is:

```
lppstat -m -e -v [host1|file1] [host2|file2]
```

I will refer to host1 as the 'target' system and host2 as the 'default' system. *lppstat* identifies three different types of discrepancy between the target system and the default system, depending on which flags are used. Any or all of these flags may be specified:

- **-v** version comparison – identify filesets that are installed on both the default and target system, but have different fileset levels.
- **-m** missing filesets – identifies filesets that are installed on the default system but are not installed on the target system.
- **-e** extra filesets – identifies filesets that are not installed on the default system but are installed on the target system.

lppstat provides multiple ways by which the target system can be compared to the default system:

1 If no parameters are specified on the command line, lppstat will compare the local host to a file called */etc/lppdefault* containing the output of the command **lslpp -Lcq > lppdefault**, which was presumably generated on some other host and copied to the local host.

2 If two hostnames are specified on the command line, they will be compared to each other directly. This requires the hosts to be on a common network and the **rsh** protocol to be enabled between them.

3 If two filenames are specified on the command line, the contents of these two files will be compared. This option allows two hosts to be compared from a remote location, or multiple hosts to be compared from a central location.

4 Any combination of hostname/filename can be specified on the command line for multiple flexibility.

On our SP systems, I typically use lppstat as follows. I first identify one node whose fileset levels I believe are current and generate the lppdefault file on this node. I then install this file as */etc/lppdefault* on all nodes using **rdist**. You may consider using **pcp** or **supper** as alternatives to **rdist**, or simply put the lppdefault file in a filesystem which is available on all hosts, eg NFS or GPFS. I then run the following command from the control workstation:

```
dsh -av lppstat -mev
```

This runs *lppstat* on all nodes and compares each node's **lpp** levels to those in the lppdefault file.

I find *lppstat* to be a useful tool on the several thousand SP nodes I administer. I believe it will prove valuable to any administrator of any size SP system or collection of stand-alone AIX hosts.

```
#!/usr/bin/perl
#
# list differences between lpp levels of two entities which can
# either be hostnames or files containing lslpp -Lcq output
#
# Usage:     lppstat -m -e -v [host1/file1] [host2/file2]
#            -m lists missing filesets from host1/file1
#            -e lists extra filesets on host1/file1
#            -v lists version mismatches
#            one or more of these flags must be specified
#            default for host1/file1 is localhost
#            default for host2/file2 is /etc/lppdefault
#
require 5.003;
use Getopt::Std;
##
## Globals
##
$path_lslpp =          "/usr/bin/lslpp";
$path_rsh =            "/usr/bin/rsh";
$path_lppdefault =     "/etc/lppdefault";
##
## MAIN
##
my ($sys, $def, $deflev, $syslev);
my (%system, %default);
# handle arguments
getopts("mev") or usage();
($opt_m || $opt_e || $opt_v) or usage();
if ($#ARGV < 0) {
      $sys = "localhost";
      $def = $path_lppdefault;
} elsif ($#ARGV == 1) {
      $sys = shift(@ARGV);
      $def = shift(@ARGV);
} else {
      usage();
}
# build lpp hash for host1/file1
lslpp(\%system,$sys);
# build lpp hash for host2/file2
lslpp(\%default,$def);
if ($opt_e) {
      foreach $key (keys %system) {
```

```
            if (!defined($default{$key})) {
                  print("Extra fileset: $key:$system{$key}\n");
            }
      }
}
if ($opt_m) {
      foreach $key (keys %default) {
            if (!defined($system{$key})) {
                  print("Missing fileset: $key:$default{$key}\n");
            }
      }
}
if ($opt_v) {
      foreach $key (keys %default) {
            if (defined $system{$key}) {
                  ($deflev) = split(/:/, $default{$key});
                  ($syslev) = split(/:/, $system{$key});
                  if ($deflev ne $syslev) {
   print("Version mismatch: $key:$system{$key} (default is $deflev)\n");
                  }
            }
      }
}
exit(0);
##
## SUBS
##
sub lslpp
{
      my ($list,$source) = @_;
      my ($fset, $lev, $x);
      if ($source eq "localhost") {
            (open(LPP, "$main::path_lslpp -Lcq |"));
      } elsif (is_file($source)) {
            (open(LPP, "< $source"));
      } elsif (is_host($source)) {
         (open(LPP, "$main::path_rsh $source $main::path_lslpp -Lcq |"))
      } else {
            print("Error: $source is not a valid host or file\n");
            exit(1);
      }
      while (<LPP>) {
            #Pkg:Fileset:Level:State:PTF Id:Fix State:Type:Desc:
            chomp;
            ($x, $fset, $lev, $x, $x, $x, $x, $x) = split(/:/);
            ${$list}{"$fset"} = $lev;
      }
      close(LPP);
}
```

```
sub is_file
{
      my ($fn) = @_;
      return -r $fn;
}
sub is_host
{
      my ($hn) = @_;
      my $dummy;
      return ($dummy = gethostbyname($hn));
}
sub usage
{
      print ("Usage: lppstat -m -e -v [host1/file1] [host2/file2]\n");
      print ("      -m lists missing filesets from host1/file1\n");
      print ("      -e lists extra filesets on host1/file1\n");
      print ("      -v lists version mismatches\n");
      print ("      one or more of these flags must be specified\n");
      print ("      default for host1/file1 is localhost\n");
      print ("      default for host2/file2 is /etc/lppdefault\n");
      print;
      exit(1);
}
```

Robin Goldstone (robing@llnl.gov)
System Administrator
Lawrence Livermore National Laboratory (USA)

E-mail alerts

Our e-mail alert service will notify you when new issues of *AIX Update* have been placed on our Web site. If you'd like to sign up, go to http://www.xephon.com/aixupdate.html and click the 'Receive an e-mail alert' link. Or, send an e-mail to majordomo@xephon.net with the words 'subscribe' and 'aix-update' in the body of the message (without the quotes).

Help for ESS fibre-attached drives

After three new IBM ESS ‘shark’ storage servers rolled in our door, it quickly became apparent that some sort of utility was needed in order to take the confusion out of matching AIX ‘hdisks’ to ESS ‘LUNs’.

BACKGROUND

To those who are not familiar with the IBM ESS, it is a very large and scalable storage server for many different operating systems and their associated hardware. It is not uncommon for a single ESS to be the storage centre for many AIX, NT, AS/400, and other Unix hosts.

Contained in the ESS is a pair of H70 RS/6000s loaded with a scaled-down version of AIX tuned especially for serving up fast disk I/O. Each H70 node contains 3GB of memory to cache I/O activity, and several fast SSA adapters for connection to the disk drives. The drives themselves are standard high-speed SSA architecture, and can be ordered in many different sizes to meet the total storage needs of the buyer. Redundancy is a high priority with RAID technology used throughout for data storage, and each hardware component being backed up by another.

Connection from a given host to the ESS can be made primarily by ultra-SCSI or fibre-channel adapters, with fibre being the faster but more expensive alternative. Each volume of storage that can be assigned to a host is assigned a ‘LUN’ or logical unit number. This LUN is used to track the volume through the Web-based interface to the ESS, known as the StorWatch Specialist.

Once a volume or LUN is assigned to an AIX host through the StorWatch Specialist, the configuration manager (cfgmgr) needs to be run to configure the new drive or hdisk. However, once this has been done for several LUNs on a given host, it becomes increasingly difficult to relate each LUN to its associated hdisk.

Essdrv.sh is a shell script that will easily take the guesswork out of

mapping ESS LUNs to AIX hdisks, and, although written primarily for fibre-attached AIX hosts, it could easily be modified for SCSI-attached and other Unix hosts.

ESSDRV.SH

```
#!/bin/ksh
###############################################################################
# /usr/local/utils/essdrv.sh
#
# by Adam Spangler
###############################################################################
# purpose: To give general information for fibre channel attached ESS
# drives.
###############################################################################
# revisions:
#
###############################################################################
clear
OUTFIL=/tmp/essdrv.out
TMPFIL=/tmp/essdrv.tmp

# Build base sort file, initialize header to out file.

lsdev -Cc disk | grep FC | grep Available | sort -tk +1 -n > $TMPFIL

printf "\n%-12s%-18s%-8s%-12s%-9s%-15s\n" "HDISK" "VG" "PVID" "ADAPTER"
"LUN ID" "SIZE meg/gig" | tee $OUTFIL
echo "-------------------------------------------------------------------
-----\n" | tee -a \ $OUTFIL

# Start loop for disks in sort file to show information for each.
# Disks that are "unreachable" are those that are in volume groups
# that are varied off at the time this script is run.
# Disks that are not "assigned" are not in a volume group yet
# therefore certain information cannot be obtained.

for DISK in `cat $TMPFIL | awk '{print $1}'`
do
   LINE=`lspv | grep "$DISK "`
   VG=`echo $LINE | awk '{print $3}'`
   PVID=`echo $LINE | awk '{print $2}' | cut -c13-16`
   ADAPTER=`lsdev -Cc disk | grep "$DISK " | awk '{print $3}'`
   LUNID=`lsattr -El $DISK | grep lun_id | awk '{print $2}' | cut -c3-6`
   SIZE_INFO=`lspv $DISK 2>/dev/null | grep TOTAL`
   if [[ `echo $SIZE_INFO | awk '{print $3}'` = "???????" ]]
   then
```

```
      SIZE_MEG="Disk is unreachable"
      SIZE_GIG=
      printf "%-12s%-18s%-8s%-12s%-8s%-10s%-10s\n" "$DISK" "$VG" "$PVID"
"$ADAPTER" \ "$LUNID" "$SIZE_MEG" | tee -a $OUTFIL
   elif [[ `echo $VG` = "None" ]]
   then
      SIZE_MEG="Disk not assigned"
      SIZE_GIG=
      printf "%-12s%-18s%-8s%-12s%-8s%-10s%-10s\n" "$DISK" "$VG" "$PVID"
"$ADAPTER" \ "$LUNID" "$SIZE_MEG" | tee -a $OUTFIL
   else
      SIZE_MEG=`lspv $DISK 2>/dev/null | grep TOTAL | awk '{print $4}' |
tr -d "("`
      SIZE_GIG=`echo $SIZE_MEG | awk '{print $1/1000}'`
      printf "%-12s%-18s%-8s%-12s%-9s%-10s%-10s\n" "$DISK" "$VG" "$PVID"
"$ADAPTER" \ "$LUNID" "$SIZE_MEG / $SIZE_GIG" | tee -a $OUTFIL
   fi
done
echo "\n\nOutfile is \"/tmp/showess.out\" if you wish to print.\n\n"
rm $TMPFIL
```

SAMPLE OUTPUT

An example of the output is shown below:

```
HDISK       VG              PVID    ADAPTER    LUN ID  SIZE meg/gig
------------------------------------------------------------------------

hdisk2     testdisk_vg    64b2    20-58-01   5600    Disk is unreachable
hdisk3     new_vg         3c29    10-58-01   5403    7624 / 7.624
hdisk4     bprod_logvg    09b9    20-58-01   5602    Disk is unreachable
hdisk5     new_vg         b29a    10-58-01   5703    7624 / 7.624
hdisk6     new_vg         98be    10-58-01   5207    7624 / 7.624
hdisk7     new_vg         81e8    10-58-01   5503    7624 / 7.624
hdisk8     new_vg         6f28    10-58-01   5007    7624 / 7.624
hdisk9     new_vg         60f9    10-58-01   5307    7624 / 7.624
hdisk10    new_vg         1cf5    10-58-01   5105    7624 / 7.624
hdisk11    new_vg         f5e2    10-58-01   5106    7624 / 7.624
hdisk12    Susans_vg      2487    20-58-01   5201    30496 / 30.496
hdisk13    Susans_vg      2624    20-58-01   5202    30496 / 30.496
```

Adam Spangler
Brinks Home Security (USA)

Enhancing the vi editor

INTRODUCTION

I'm a fan of the **vi** editor, but I've found a few frustrations when using it on a day-to-day basis. I have developed a few tools related to **vi**. Some of these are useful especially in a software development environment, eg when writing C programs. Others are for general use.

For scripts *vir* and *vib* to operate correctly, a change to the login script, ie *.profile*, is required. For example, add the following line to *.profile*:

```
alias vi=vir
```

Here's a summary of the scripts, covered in detail below:

- *vib* – creates a back-up file automatically allowing changes to be 'undone'.
- *vir* – warns if a file is read-only before going into **vi**.
- *vix* – used to create a script file, setting the execute flag.
- *vis* – variation on *vix*, allowing a file to be edited from a user's bin directory.

IMPLEMENTATION NOTES

I'm staying with the 'self-commenting' method of writing scripts that I've used in previous articles. Whilst it takes a little more effort when writing a script, it makes the script so much more readable a number of years later.

SCRIPTS

vib

A feature I miss with **vi** that I took for granted with editors in the past is the ability to automatically create a back-up of the previous file. This allows mistakes regretted after saving the file to be automatically

'undone' by renaming the *.bak* file to the current spoiled file. I recommend placing an alias command in the user's login script as described above to ensure this script is run whenever the user executes the **vi** command.

```
#!/bin/sh
#   vib
#   ---
#   An enhancement to the vi editor to create a back-up of the
#   last file edited, thus allowing a kind of 'undo' facility
#   If a file called john.text is edited using the command:
#            vib john.text
#   ... a file called john.text.bak will be created containing the
#   previous version of the file.  Much of the complexity of this
#   script is to ensure the current '.bak' version of the file is only
#   overwritten if the file being edited is changed - ie if the user
#   "quits" vi, there is no need to overwrite any existing 'previous
#   version' of the file.
#   This file should be placed in $HOME/bin, PATH set to include $HOME/
#   bin and an alias command added to the login profile, eg:
#            alias vi=vir
#
#   Implementation notes
#   --------------------
#   1.  The DO_main function is used to make the script more readable
#
DO_main ()
{
      PrintUsage $1
      SaveFileToTemporaryFile $1
      EditFile $1             # Call vi
      if FileIsUnchanged $1
      then
            RemoveTemporaryFile $1
      else
            CopySavedTemporaryFileTo_BAK_File $1
      fi
}
PrintUsage ()
{
      [ "$1" = "" ] && echo "Usage: $Ø filename" && exit
}
SaveFileToTemporaryFile ()
{
      TFILE=/tmp/vib$$
      cp $1 $TFILE
}
EditFile ()
{
```

```
      vi $1
}
FileIsUnchanged ()
{
      diff $1 $TFILE >/dev/null
}
CopySavedTemporaryFileTo_BAK_File ()
{
      mv -f $TFILE $1.bak
}
RemoveTemporaryFile ()
{
         rm $TFILE
}
# Do not remove the DO_Main call ...
DO_main $*
exit Ø
```

vir

When doing some serious software development using C under a source code control system such as *sccs*, the programmer must remember to check out a file for edit before editing. If a file is edited without remembering to do this, the file will be read only. **Vi** has the annoying feature of allowing an edit to take place, even though a file is actually read only. OK, it's fair to say that **vi** displays a warning at the bottom of the screen, but it's hard to notice and means you're already in **vi** so have to, at the very least, quit.

This script checks whether a file is read-only, displays a message if it is, and gives the user a chance to cancel, eg ctrl-C, the script before going into **vi**. For good measure, it also checks for the existence of a file and again warns before going into **vi**. This feature is useful if a filename is mistyped.

This last feature may seem like a trivial enhancement to **vi**. When I'm 'in the groove', programming away like crazy, I've found it much less frustrating to hit ctrl-C if I mistype a filename than to be in **vi** with a blank screen and have to quit out again.

```
#!/bin/sh
#   vir
#   ---
#   An enhancement to the vi editor to check whether a file is
#   read-only BEFORE going into vi.  When using a source code control
```

```
#   system such as sccs, eg when working in the C programming language,
#   it's quite easy to edit a whole file without realizing that vi is
#   unable to write back the end result.
#   A warning is printed.  The user should press RETURN to continue, or
#   cancel with CTRL-C.
#   This file should be placed in $HOME/bin, PATH set to include $HOME/
#   bin and an alias command added to the login profile, eg:
#            alias vi=vir
#
#   Implementation notes
#   --------------------
#   1.    The DO_main function is used to make the script more readable
#
DO_main ()
{
      PrintUsage $1
      if FileNotFound $1
      then
            echo "$1 not found"
            sleep 1
            read x
      elif FileReadOnly $1
      then
            echo "!!!!! $1 read only !!!!!"
            sleep 1
            read x
      fi
EditFile $*
}
PrintUsage ()
{
      [ "$1" = "" ] && echo "Usage: $Ø filename" && exit
}
EditFile ()
{
      vi $*
}
FileNotFound ()
{
      [ ! -r "$1" ]
}
FileReadOnly ()
{
      [ ! -w "$1" ]
}
# Do not remove the DO_Main call ...
DO_main $*
exit Ø
```

vis

When the current directory is sitting somewhere deep in a source code tree, I've found it useful to be able to edit scripts directly in my personal bin directory. With this script, if I'm creating a new script, I stay in the current directory and type vis [scriptname].

```
#!/bin/sh
#   vis
#   ---
#   Allows a script to be edited from a user's bin directory
#   regardless of current directory
#   This file should be placed in $HOME/bin,
#                                          PATH set to include $HOME/bin
#
#   Implementation notes
#   --------------------
#   1.     The DO_main function is used to make the script more readable
#
DO_main ()
{
      PrintUsage $1
      GetFullPathNameOfScript $1
      EditScript
}
PrintUsage ()
{
      [ "$1" = "" ] && echo "Usage: $Ø filename" && exit
}
GetFullPathNameOfScript ()
{
      FULLPATHNAMEOFSCRIPT=`whence $1`
}
EditScript ()
{
      vi $FULLPATHNAMEOFSCRIPT
}
# Do not remove the DO_Main call ...
DO_main $*
exit Ø
```

vix

Creating new scripts is a two-step process:

1 **vi** the script.

2 Use *chmod* to set the execute bit.

This script essentially combines these two steps.

This may seem trivial, but in reality the two apparent steps are normally more like four. They are:

1 **vi** the script.

2 Execute the script.

3 Realize that you've forgotton to do the *chmod* command.

4 Use *chmod* to set the execute bit.

```
#!/bin/sh
#   vix
#   ---
#   Used to create a new script file, setting the execute bit after the
#   file has been created using vi
#
#   This file should be placed in $HOME/bin,
#                                          PATH set to include $HOME/bin
#
#   Implementation notes
#   --------------------
#   1.    The DO_main function is used to make the script more readable
#
DO_main ()
{
      PrintUsage $1
      EditFile $1
      MakeScriptExecutable $1
}
PrintUsage ()
{
      [ "$1" = "" ] && echo "Usage: $Ø filename" && exit
}
EditFile ()
{
      vi $1
}
MakeScriptExecutable ()
{
      chmod +x $1
}
# Do not remove the DO_Main call ...
DO_main $*
exit Ø
```

John Rainford (UK) © Xephon 2001

Process identification utility

Often you may notice a process run on your system, which for some reason or other you wish to identify. It may be a process that is using large amounts of CPU, or just a process you have not noticed before and you wonder where it came from.

The pidTree script, when supplied with a process ID, displays all its parent and child processes, in order to give you some clue as to where the process in question came from. This is achieved by processing the output of the **ps** command using an **awk** command, as shown below:

```
#!/usr/bin/ksh
#
# Script: pidTree
# Author: Roger Wickings
# Aim:    Display parents and children of a specified process ID
#

awk="/usr/bin/awk"
basename="/usr/bin/basename"
cat="/usr/bin/cat"
cut="/usr/bin/cut"
id="/usr/bin/id"
ps="/usr/bin/ps"
rm="/usr/bin/rm"
sed="/usr/bin/sed"
tr="/usr/bin/tr"

TMPDIR="/tmp"

# functions

cleanUp()
{
  $rm -f $PSFILE 2>/dev/null
  return
}

initialization()
{
  if test "$TARGETS" = ""
  then
    TARGETS="1"
  fi
```

```
  HIGHLIGHT="yes"
  new_targets=""
  for target in $TARGETS
  do
    if test "$target" = "-n"
    then
      HIGHLIGHT="no"
    else
      new_targets="$new_targets $target"
    fi
  done
  TARGETS=`echo "$new_targets" | $sed "s/[^0-9 ]//g" `

  USER=`$id | $tr "()" "  " | $awk '{print $2}' `
  PSFILE="$TMPDIR/$SCRIPT.$USER.$$.ps"
  $ps -Ao pid,ppid,user,etime,time,vsz,tty,args |
    $cut -c1-100 > $PSFILE
  return
}

listPids()
{
  echo
  for target in $TARGETS
  do
    (
      echo "PARM T $target"
      echo "PARM H $HIGHLIGHT"
      $cat $PSFILE
     ) |
      $awk 'BEGIN {
               psub = 0
              }
        $1 == "PARM" {
               if ( $2 == "T" ) { target    = $3 }
               if ( $2 == "H" ) { highlight = $3 }
              }
        $1 != "PARM" {
               psub += 1
               pid[psub]  = $1
               ppid[psub] = $2
               line[psub] = $0
              }
        END   {
                maxsub = psub

                parent = target               ##### find parents #####
                pntsub = 0
```

```
  olevel = 0
  while ( parent > 0 )
  {
    for ( fsub = 1 ; fsub <= maxsub ; fsub++ )
    {
      if ( pid[fsub] == parent )
      {
        pntsub++
        pline[pntsub] = line[fsub]
        parent = ppid[fsub]
        break
      }
    }
    if ( fsub > maxsub )
    {
      break
    }
  }
  if ( pntsub == 0 )
  {
if ( highlight == "yes" ) { system("/usr/bin/tput smso") }
print "Process id", target, "not found."
if ( highlight == "yes" ) { system("/usr/bin/tput rmso") }
    exit
  }
  else
  {
    print "Process id", target
  }
  for ( ; pntsub > 1 ; pntsub-- )
  {
    olevel++
    printf "%-2s %s\n", olevel, pline[pntsub]
  }

  olevel++
if ( highlight == "yes" ) { system("/usr/bin/tput smso") }
printf "%-2s %s\n", olevel, pline[1]
if ( highlight == "yes" ) { system("/usr/bin/tput rmso") }

  level   = 2                  ##### find children #####
  isub[1] = 0
  itar[1] = target
  csub    = maxsub
  while ( level > 0 )
  {
    csub++
    if ( csub > maxsub )
    {
      level--
```

```
                    if ( level == 0 )
                    {
                      break
                    }
                    csub = isub[level]
                    ctar = itar[level]
                  }
                  else
                  {
                    if ( ppid[csub] == ctar && pid[csub] != ctar )
                    {
                      printf "%-2s %s\n", olevel + level, line[csub]
                      isub[level] = csub
                      level++
                      isub[level] = 0
                      itar[level] = pid[csub]
                      ctar        = pid[csub]
                      csub        = 0
                    }
                  }
                }
              }'
    echo
  done
  return
}

# start of main processing

SCRIPT=`$basename $0 `
TARGETS=`echo "$*" | $sed "s/\"//g" `

initialization
listPids
cleanUp

exit 0
```

The script is executed as follows:

```
pidTree 16584
```

which produces the following output:

```
Process id 16584
1       1      0     root 95-19:50:06    16:55:23   688      - /etc/init
2    3642      1     root 95-19:48:38    00:00:05   392      - /usr/sbin/srcmstr
3   12822   3642     root 95-19:44:51    00:00:07   512      - /usr/dt/bin/dtlogin
4   91334  12822     root 11-22:25:22    00:00:01   748      - dtlogin <ops_pc:0>
5   16584  91334 operator 11-22:22:50    00:00:00   828      - /usr/dt/bin/dtsession
6   34130  16584 operator 11-22:22:41    00:00:55  1040      - /usr/dt/bin/dtterm
```

```
7   45092  34130 operator 11-22:22:39    00:00:00   320  pts/8 /bin/ksh
8   35604  45092 operator  2-00:15:43    00:00:31   528  pts/8 telnet gp001ukb
6   36078  16584 operator 11-22:22:41    00:00:00  1036      - /usr/dt/bin/dtterm
7   94208  36078 operator 11-22:22:39    00:00:00   320 pts/12 /bin/ksh
8   40842  94208 operator  2-00:14:47    00:00:00   524 pts/12 telnet gp001uka
```

By default the line displaying the target process is highlighted, but this can be suppressed by specifying the **-n** option, eg *pidTree -n 16584*.

Additionally, pidTree will accept multiple pids and display the appropriate tree for each pid specified, eg *pidTree 16584 35604 94208*.

Roger Wickings
Systems Programmer
FT Interactive Data (UK)

Standby system failover – part 2

This month we conclude the code to set up the standby failover system.

F50CUTOVER.SH

```
#!/bin/ksh
set -v
# F50cutover.sh
#=================================================================#
#   DESCRIPTION:                                                  #
#   -----------                                                   #
#   The initial presumption here is that this primary F50         #
#   is still running and able to respond enough to run this       #
#   script. If not, it should be shutoff and the network cable    #
#   removed. Then proceed with script 570cutover.sh to make the   #
#   standby 570 takeover the role of the primary system.          #
#                                                                 #
#   This will change the F50 into a passive system, giving        #
#   up its application(s). Its hostname will become               #
#   <hostname>_temp and its IP address will become                #
#   xx.xxx.xxx.239. (If you already use .239, choose some         #
#   other unused IP ending number.                                #
#                                                                 #
```

```
#   INSTRUCTIONS FOR USE:                                              #
#   ---------------------                                              #
#   Contact the remote site and have the users logout.                 #
#   Telnet into the remote F5Ø production machine, and then            #
#   execute this script using the root account, as follows:            #
#         /F5Øcutover.sh                                               #
#                                                                      #
#   NOTE:                                                              #
#   ----                                                               #
#   Once the IP address gets changed in step CHANGE_IP,                #
#   you will be disconnected from the site. You must then              #
#   reconnect using the xx.xxx.xxx.239 address.                        #
#   Once reconnected, execute this script passing                      #
#   the step-argument as follows:                                      #
#         /F5Øcutover.sh  RENAME_CRONS                                 #
#                                                                      #
#   This will resume the processing after the IP change.               #
#====================================================================#
date
if [ $1 != "" ]            #if no argument is passed, assume step 1
then
    STEP=$1
else
    STEP="SHUTDOWN_SNA"
fi
#============================================================#
date
# This sample site has an SNA connection to a remote mainframe. So
# the first thing we want to do is shutdown the SNA sessions.
if [ ${STEP} = "SHUTDOWN_SNA" ]
then
    /SYSMGR/sna_shutdown
    STEP="CHANGE_IP"
fi
#============================================================#
date
# Here we are changing this F5Ø's hostname and IP address to
# temporary values. Note that it is ESSENTIAL that this command
# is 'nohupped' in the background (&). Once this step executes,
# your remote Telnet session to the site will hang and you will
# have to login again in a few moments to the "239" address to
# continue this processing.
if [ ${STEP} = "CHANGE_IP" ]
then
   set +v
   echo "Detaching the network interface..you will be disconnected.."
   echo "In a few moments, Telnet in again to the '239' temp IP address"
   echo "and then execute the following command:"
   echo "/F5Øcutover.sh  RENAME_CRONS"
   echo ""
```

```
   nohup ksh /chg12to239  &
   sleep 10
   exit 0
   STEP="RENAME_CRONS"
fi
#============================================================#
date
# This is the step that you will start with when you log back
# in remotely using the "239" address. See the instructions at
# the beginning of this script for further details.
if [ ${STEP} = "RENAME_CRONS" ]           #save the production cron jobs
then
    mv /var/spool/cron/crontabs/root
                                     /var/spool/cron/crontabs/root_save
    mv /var/spool/cron/crontabs/some_account
                               /var/spool/cron/crontabs/some_account_save
#...Now refresh the cron daemon...
    pidcrond=$(ps -aef | grep -v grep | grep "cron" |awk '{print $2}')
    kill -9 ${pidcrond}
    STEP="KILL_MISC"
fi
#============================================================#
date
# In this step we are killing any known running jobs that would be
# holding open any files in filesystems that we later intend to
# dismount. In other words, we want to avoid the message that says
# 'filesystem is busy' when we try to unmount them.
if [ ${STEP} = "KILL_MISC" ]
then
    fuser -k /dev/some_app_LV        # -k should kill all the PIDs found
    umount /some_app_filesystem      #(see the associated LV above)
    if [ $? -ne 0 ]
    then
      set +v
        echo "Could not unmount the /some_app_filesystem filesystem."
        echo "Please fix this and restart from step KILL_MISC"
      echo "ie /F50cutover.sh  KILL_MISC"
        echo "Terminating procedure..."
        exit 1
    fi
    STEP="UNMOUNT_ALL"
fi
#============================================================#
date
#...Now we attempt to unmount all filesystems (except for the normally
#...busy system filesystems like /, /usr, /tmp, and so on.
if [ ${STEP} = "UNMOUNT_ALL" ]
then
    umount -a
    STEP="VARY_OFF"
```

```
fi
#============================================================#
date
if [ ${STEP} = "VARY_OFF" ]
then
    varyoffvg datavg
    if [ $? -ne Ø ]
    then
      set +v
        echo "Could not varyoff VG datavg."
        echo "Please fix this and restart from step VARY_OFF"
      echo "i.e.  /F5Øcutover VARY_OFF"
        echo "Terminating procedure..."
        exit 1
    fi
    STEP="EXPORT_VG"
fi
#============================================================#
date
#...Now we give up the application vg by exporting it. This
#...will allow it to be taken over by the standby machine later.
if [ ${STEP} = "EXPORT_VG" ]
then
    exportvg datavg
    if [ $? -ne Ø ]
    then
      set +v
        echo "Could not export VG datavg."
        echo "Please fix this and restart from step EXPORT_VG"
      echo "ie  /F5Øcutover.sh  EXPORT_VG"
        echo "Terminating procedure..."
        exit 1
    fi
fi
#============================================================#
date
set +v
echo "The F5Øcutover procedure is now COMPLETE."
echo "You should now proceed with the 57Øcutover.sh"
echo "script on the standby machine to make it the new live machine."
#end of job.
```

570CUTOVER.SH

```
#!/bin/ksh
set -v
# 57Øcutover.sh
#================================================================#
#   DESCRIPTION:                                                 #
```

```
#   -----------                                                    #
#   This will change the standby 57Ø system into the F5Ø system #
#   Its hostname will become the F5Ø's normal hostname, and its #
#   IP address will become xx.xxx.xxx.12                           #
#                                                                  #
#   INSTRUCTIONS FOR USE:                                          #
#   --------------------                                           #
#   Telnet into the standby machine, and execute this script     #
#   using the root account, as follows:                            #
#         /57Øcutover.sh                                           #
#                                                                  #
#   NOTE:                                                          #
#   ----                                                           #
#   Once the IP address gets changed in STEP=CHANGE_IP, you      #
#   will be disconnected from the standby machine. You must      #
#   then reconnect using the xx.xxx.xxx.12 primary ip_address.   #
#                                                                  #
#   Once reconnected, execute this script passing the            #
#   step-argument as follows:                                      #
#         /57Øcutover.sh  IMPORT_VG                                #
#                                                                  #
#   This will resume the processing after the IP change.         #
#================================================================#
date
if [[ $1 != "" ]]      #if no argument specified, assume step 1
then
    STEP=$1
else
    STEP="SWITCH_AB"
fi
#============================================================#
date
#...In this optional site-specific step, we want to ensure that the
#...remote staff person gives us a hand by switching the A/B box to
#...the "B"(57Ø) position, for the purpose of establishing the SNA to
#...remote mainframe connection required for the particular application
#...that runs on this machine.
if [ ${STEP} = "SWITCH_AB" ]
then
    set +v
    echo "Please have the remote staff person switch the A/B box to the
57Ø position now."
    echo "\nHas this has been done? (y/n) \c"
    read answer overflow
    case ${answer} in
       "y"|"Y")
                echo "Proceeding to switch the IP address..."
        ;;
       *)
                echo "Exiting procedure. Please rerun this when you are"
```

```
                echo "prepared to switch the A/B box."
                exit 1
        ;;
    esac
    STEP="CHANGE_IP"
fi
#============================================================#
date
      set -v
# Here we are changing this standby machine's hostname and
# IP address to that of the production system.
# Note that it is ESSENTIAL that this command is 'nohupped' in
# the background (&).
# Once this step executes, your remote Telnet session to the
# site will hang, and you will have to login again in a few
# moments to the "12" address to continue this processing.
if [ ${STEP} = "CHANGE_IP" ]
then
    set +v
    echo "Detaching the network interface..you will be disconnected.."
    echo "In a few moments, please log back in again using the"
    echo "production system's IP address, and the execute this command:"
    echo "/57Øcutover  IMPORT_VG"
    echo ""
    nohup ksh /chg13to12  &

    sleep 1Ø            #let the IP address change
    exit Ø              #get out (only to log back in moments later)
    STEP="IMPORT_VG"
fi
#============================================================#
date
set -v
#...In this step, we are now attempting to takeover the external
#...disk data volume group normally owned by the production F5Ø
#...machine. See the ImportVG script for further comments.
if [ ${STEP} = "IMPORT_VG" ]
then
    /ImportVG
    STEP="START_SNA"
fi
#============================================================#
date
#...Here we are restarting the SNA sessions that this
#...particular production machine requires for its application.
if [ ${STEP} = "START_SNA" ]
then
    /MBSYSMGR/restart_sna
    STEP="RENAME_CRONS"
fi
```

```
#============================================================#
date
#...Finally, we are activating the production cron jobs
#...as required by the production application(s). These cron
#...files are actually copied overnight from the real
#...production machine (see script box_to_box.sh) for further
#...details on this.
#...First we will save the standby system's cron files...
if [ ${STEP} = "RENAME_CRONS" ]
then
      mv /var/spool/cron/crontabs/root   /var/spool/cron/crontabs/
570_root
mv /var/spool/cron/crontabs/some_account \
                              /var/spool/cron/crontabs/570_some_account
#...Now activate the production crons...
      mv /var/spool/cron/crontabs/F50_root
                                          /var/spool/cron/crontabs/root
      mv /var/spool/cron/crontabs/F50_some_account   \
            /var/spool/cron/crontabs/some_account
#...Now refresh the cron daemon...
      pidcrond=$(ps -aef | grep -v grep | grep "cron" |awk '{print $2}')
      kill -9 ${pidcrond}
fi
#============================================================#
date
# At this time, the users should be able to reconnect to what
# appears to be the normal production system again. In fact they
# will now be running on the standby 570 machine but it will be
# transparent to them.
set +v
#(end of job).
```

Michael G Stanton
Supervisor Mid-range Systems
Mercedes-Benz (USA)

An automated file transfer system for AIX

INTRODUCTION

The information technology industry is in a constant state of flux. From host-centric systems we evolved to client/server computing. With the advent of the Web, the focus has now shifted to *n*-tiered

architectures. Applications are presented on GUI clients served by application servers. These application servers in turn access data from database servers. Other middleware gateways are also used for different functions.

With the distributed server architecture, there never has been a greater need for moving data between multiple servers on a regular basis. Some examples include consolidating log files, moving application files to multiple application servers, centralizing errors to a common server, etc.

The underlying protocol that makes this happen is the good old File Transfer Protocol (FTP). A number of commercial packages have been written around FTP, which can automate the task of transporting files to different servers.

We solved this issue of moving data among different AIX servers by writing a simple shell-based application called *FTPTAB*. The system is completely written for use with the Korn shell and is very easily ported to different versions of AIX. It has not been tested on other flavours of Unix, but should not be too difficult as long as the Korn shell is available.

The FTPTAB system provides the following features:

1 Job scheduling (daily, weekly, and monthly).

2 File transport (*Gets* and *Puts*).

3 Error logging.

4 Multiple threads based on individual login.

5 Pre- and post-processing.

6 ASCII and BINARY options.

7 Retry logic with timeouts.

SETTING UP AND USING FTPTAB

To set up the FTPTAB system, create an AIX login called ftpuser with its home directory set to */usr/ftptab*. Create the following directories under */usr/ftptab*:

- run
- logs
- in
- out

Extract the scripts from this article into the */usr/ftptab/run* directory and make sure that this directory is placed in your path. Please preserve the file names.

Set up a **cron** entry for the ftpuser to run the FTPTAB system every 10 minutes. Here is a sample *cron* entry:

```
0,10,20,30,40,50 * * * * ; cd /usr/ftptab/run ; ksh ftptab.sh
```

If you will need to control small and frequent FTP jobs then you can set the **cron** to activate this script every minute. If you do so, change the **cron** window variable in the *ftptab_prof.sh* script.

For each FTP job, create an entry in the configuration file (*ftptab_ftpuser.f*). Each job must be on a separate line. Here is a sample:

```
W:2::1:30:10.10.2.4:ftptab_user:ftptab_pwd:P:/usr/ftptab/in:*.log:/usr/
ftptab/out:B:ftptab_pre.sh::3:300
```

This job reads as follows:

It will be executed every week (w) on Tuesdays (2) at 1:30 am. It will contact the host 10.10.2.4 with the userid (ftptab_user) and password (ftptab_pwd). It will take *.log files from */user/ftptab/out* and put (P) them in */usr/ftptab/in* at the remote end. It will use the binary mode (B) for the file transfer. Before running the actual FTP, it will also run a preprocessing script (ftptab_pre.sh). This job will be attempted at most 3 times. Each time it will try for up to 300 seconds before timing out.

You can add as many entries in the configuration file as you need. Please note that the system will single-thread through this file per user. If this causes time conflicts with multiple jobs, you can run this system under multiple logins because it always looks at the configuration file for $LOGNAME.

UNDERSTANDING FTPTAB

The **cron** is the underlying scheduler used for this system. At each **cron** invocation, the script *ftptab.sh* is executed. The script identifies the configuration file for the current user and then processes it one job at a time. If the execution schedule for the job matches the current date and time within the **cron** window then the job is passed on to the *ftptab_driver.sh* script for execution.

The *ftptab_driver.sh* script runs any preprocessing scripts identified by the job. An example of pre-processing could be to compress files before sending, to save on transmission time.

It then builds a *.netrc* file in the user's home directory by calling the *bld_netrc.sh* script. FTP expects that the *.netrc* file will be placed in the user's home directory to avoid prompting for userid and password during the actual FTP session.

The actual FTP is then invoked by calling the *ftptab_run.sh* script. This script is executed in the background so that the parent script can monitor its execution. If the *ftptab_run.sh* does not finish within the timeout period then the process is killed. If the retry count is higher than 1 then the job is restarted. This avoids the problem caused by hung FTP jobs. Do make sure that the timeout interval is set appropriately, based on the amount of data being transmitted and the available bandwidth between the two servers.

Upon successful completion of the *ftptab_run.sh*, a post-processing script is executed if listed in the configuration file.

TROUBLESHOOTING FTPTAB

It is best to try a couple of monitored runs before placing this system into production use. You should also get familiar with the configuration file and each parameter listed.

The best place to troubleshoot any problems is to look at the logs generated in the 'logs' directory. The logs are not too user-friendly but do contain a lot of information about each run.

FUTURE DIRECTIONS

Work is currently in progress to add a simple user interface to manage the configuration file. We are also working on improving the error logging.

FTPTAB_FTPUSER.F

```
#####################################################################
#job_type:dow:dom:hour:min:rhost:ruser:rpassword:transfer:rdir:
(continued ..)
#files:ldir:mode:prescript:postscript:retry:safety
# There should be a total of 17 fields ( : separated )
#
# job_type=D (Daily) | W (Weekly) |  M (Monthly)
# dow=Day of Week (Sunday Ø - Saturday 6)
# dom=Day of Month (1 - 31)
# hour=ØØ-23
# min=ØØ-59
# rhost=IP address or resolvable hostname of remote host
# ruser=Remote User Id
# rpassword=Remote Password
# transfer=G|P|GD  (get, put, get and remote delete. GD must not be
listed as DG
# rdir=Remote directory
# files=Filename (get or put) does support wildcards
# ldir=Local directory
# mode=A|B (Ascii or Binary)
# prescript=
# postscript=
# retry=# of times to try (1-3)
# safety=# of seconds to sleep before retry
#
#  Run awk -F: '{print NF}' to find the number of fields
W:2::1:3Ø:1Ø.1Ø.2.4:ftptab_user:ftptab_pwd:P:/usr/ftptab/in:*.log:/usr/
ftptab/out:B:/usr/ftptab/scripts/ftptab.sh::3:36ØØ
#####################################################################
```

FTPTAB_PROF.SH

```
#####################################################################
function ftptab_debug
{
    if  [[ -n "${FTPTAB_DEBUG}" ]]
    then
        print "$*" >> $tracefile
    fi
}
```

```
FTPTAB_DEBUG=ON                  # uncomment for debugging
export self=ftptab.$LOGNAME
export tracefile=/usr/ftptab/logs/$self.tr
export logfile=/usr/ftptab/logs/$self.log
export flistfile=/usr/ftptab/logs/$self.rmtfiles
export statusfile=/usr/ftptab/logs/$self.status
cron_window=10
# Note: Frequency of cron in minutes (should match exactly)
#########################################################################
```

FTBTAB.SH

```
#!/bin/ksh
#########################################################################
######################
# Set up Variables #
######################
. ftptab_prof.sh
#####################################
# Function checks                 #
#####################################
#################################
# compares day of week      #
#################################
function weekly_check_func
{
  if [ "$current_dow"  -eq  "$active_dow" ]
  then
      ftptab_debug "WEEKLY MATCH"
      daily_check_func
      return $?
  else
      return 1
  fi
}
#################################
# compares date of month    #
#################################
function monthly_check_func
{
  if [ "$current_dom"  -eq  "$active_dom" ]
  then
      ftptab_debug "MONTHLY MATCH"
      daily_check_func
      return $?
  else
      return 1
  fi
}
```

```
#######################################
# Converts hours and minutes into   #
# total time in minutes. Compares   #
# total curent time to total active #
# time and if less than variable or #
# equal to zero, then return Ø .    #
#######################################
function daily_check_func
{
    (( total_curr_minutes = ( $current_hod * 6Ø ) + $current_moh ))
    (( total_active_minutes = ( $active_hod * 6Ø ) + $active_moh ))
   (( minute_difference = $total_curr_minutes - $total_active_minutes ))
    ftptab_debug "minute diff = $minute_difference"

    if  [[ $minute_difference -lt $cron_window && $minute_difference -ge
Ø ]]
    then
        ftptab_debug "DAILY MATCH"
        return Ø
    else
        return 1
    fi
}
#######################################
# Get the current date/time parms   #
#######################################
function current_datetime
{
    current_dow=`date +%w`
                        # current day of week (sunday - saturday) (Ø-6)
    current_dom=`date +%d`  # current day of month
    current_hod=`date +%H`  # current hour of day
    current_moh=`date +%M`  # current minute of hour
    export current_dow current_dom current_hod current_moh
}
#####################################
# MAIN SCRIPT                       #
# Get current Date information   #
#####################################
###############################################
# save last 3 trace files  Just in case #
###############################################
mv $tracefile.2 $tracefile.3 >/dev/null 2>&1
mv $tracefile.1 $tracefile.2 >/dev/null 2>&1
mv $tracefile $tracefile.1   >/dev/null 2>&1
#################################
# Get date and time info from #
# file for comparison         #
#################################
cat ftptab_$LOGNAME.f |       \
```

```
      grep -v "^#" |       \
      grep -v "^$" |       \
      while read row
do
    IFS=:
    set - $row
    IFS="       "
    if  [[ $# -ne 17 ]]
    then
      print "$self: Invalid Row '$row' (`date`)" >> $tracefile
      continue
    fi
    current_datetime            # get the current datetime
    jobtype=`echo $row  | awk -F: ' { print $1 } '`
    active_dow=`echo $row | awk -F: ' { print $2 } '`
    active_dom=`echo $row | awk -F: ' { print $3 } '`
    active_hod=`echo $row | awk -F: ' { print $4 } '`
    active_moh=`echo $row | awk -F: ' { print $5 } '`
    argrow=`echo $row | cut -d: -f6-`
    ####################################
    # See if the FTP job needs to be #
    # run.                           #
    ####################################
    case $jobtype in
       D) daily_check_func
        if  [[ $? -ne 0 ]]
        then
            continue
        fi
        ;;
       W) weekly_check_func
        if  [[ $? -ne 0 ]]
        then
            continue
        fi
        ;;
       M) monthly_check_func
        if  [[ $? -ne 0 ]]
        then
            continue
        fi
        ;;
       *) print -u2
        print -u2 " `date` : Invalid Job Type " >> $tracefile
        print -u2 " $row " >> $tracefile
        print -u2
        continue
        ;;
    esac
    ################################################
```

```
    # Got here means that the JOB needs to be run #
    ##############################################
    ftptab_driver.sh $argrow 1>>$logfile 2>&1
done
#########################################################################
```

FTBTAB_DRIVER.SH

```
#!/bin/ksh
#########################################################################
argrow=$1
. ftptab_prof.sh
print "\n\nFTPTAB_DRIVER_START: `date`\n\t'$argrow'\n"
#################################
# Parse out needed variables #
#################################
rhosts=`echo $argrow | cut -d: -f1 `
prescript=`echo $argrow | cut -d: -f9 `
postscript=`echo $argrow | cut -d: -f10 `
retry=`echo $argrow | cut -d: -f 11 `
safety=`echo $argrow | cut -d: -f12 `
#################################
# Execute PRESCRIPT          #
#################################
if  [[ -s "$prescript" ]]
then
    print "\trunning '$prescript' ...\n"
    $prescript
    if  [[ $? -ne 0 ]]
    then
      print "$prescript FAILED !! Will not execute FTPTAB ..."
      exit 1
    fi
    print "\tDone.\n"
fi
#########################
# Call script to build #
# netrc file           #
#########################
print "\trunning 'bld_netrc.sh $argrow' ... \n"
bld_netrc.sh $argrow
if [ $? -ne 0 ]
then
    print -u2
    print -u2 "bld_netrc.sh FAILED !!"
    print -u2
    exit 1
fi
print "\tDone.\n"
```

```
print "\tNETRC:\n`cat $HOME/.netrc`\n\n"
################################
# Run the FTPTAB             #
################################
while (( retry > 0 ))
do
    print "\t.RETRY=$retry\n\trunning 'ftptab_run.sh $rhosts &' ...\n"
    echo "99" > $statusfile             # initialize to FAILED status
    sessionsafety=$safety
    ftptab_run.sh $rhosts &
    childpid=$!
    print "\trunning 'sleep $sessionsafety' ...\n"
    ################################
    # Wake up every 60 seconds
    # and check if the ftp process
    # is still running.
    # Exit if not running or if
    # if sessionsafety expires.
    ################################
    while (( sessionsafety > 0 ))
    do
        ps -fp $childpid
        if  [[ $? -ne 0 ]]         # the process does not exist so break
        then
            break
        fi
        sleep 60
        (( sessionsafety = sessionsafety - 60 ))
        print "\tsessionsafety=$sessionsafety"
    done
    ################################
    # if sessionsafety expired, then
    # kill the ftp job because it
    # could be hung.
    ################################
    if  [[ $sessionsafety -le 0 ]]
    then
print "\tSafety expired. killing $childpid ....\n\t`ps -fp $childpid`\n"
        kill -9 $childpid
    fi
    ################################
    # Verify ftp status. If failed
    # then try again.
    ################################
    status=`cat $statusfile`
    print "FTPTAB_DRIVER: FTPTAB_RUN STATUS = $status"
    if  [[ $status -eq 0 ]]
    then
        break
    fi
```

```
    (( retry = retry - 1 ))
done
################################
# Cleanup .netrc               #
################################
print "\tRemoving HOME/.netrc ..."
rm $HOME/.netrc
print "\tDone.\n"
################################
# Execute POSTSCRIPT           #
################################
if  [[ $status -eq 0 && -s "$postscript" ]]
then
    print "\trunning '$postscript' ...\n"
    $postscript
    if  [[ $? -ne 0 ]]
    then
      print "$postscript FAILED !! "
      exit 1
    fi
    print "\tDone.\n"
fi
print "FTPTAB_DRIVER_END: `date`\n\n"
exit 0
#############################################################################
```

BLD_NETRC.SH

```
#!/bin/ksh
#############################################################################
#########################
# Set up variables     #
#########################
. ftptab_prof.sh
typeset -u transfer
typeset -u filetype
netrc=$HOME/.netrc
row="$1"                          # field 6 on from FTPTAB file
ftptab_debug "row='$row'"
IFS=:
set - $row
IFS="
"
rhost=$1
ruser=$2
rpasswd=$3
transfer=$4
rdir=$5
files=$6
```

```
ldir=$7
filemode=$8
############################
# Set transfer variable #
############################
if [ "$transfer" = "G" ]
then
    transport=mget
else
    transport=mput
fi
############################
#     Set file type     #
# Change var name due to#
# typeset to upper case #
############################
if [ "$filemode" = "B" ]
then
    filemode=binary
else
    filemode=ascii
fi
#########################################################
# Create a netrc file in users home directory  #
# with the correct parameters                  #
#########################################################
> $netrc
echo "machine $rhost" >> $netrc
echo "login $ruser" >> $netrc
echo "password $rpasswd" >> $netrc
echo "macdef init" >> $netrc
echo "cd $rdir" >> $netrc
echo "lcd $ldir" >> $netrc
echo "$filemode "  >>$netrc
echo "$transport $files"  >>$netrc
echo "quit" >> $netrc
echo "\n"  >> $netrc
exit $?
#########################################################################
```

FTBTAB_RUN.SH

```
#!/bin/ksh
#########################################################################
# Script to run .netrc file
rhosts=$1
. ftptab_prof.sh
cd $HOME
chmod 600 .netrc
```

```
ftp -i $rhosts  1>$logfile.$$ 2>&1
rc1=$?
echo "FTPTAB_RUN: FTP STATUS (rc1)=$rc1"
########################################
#Check for FTP success
########################################
# get the remote file count
remotefiles=`cat $flistfile | wc -l`
echo " Logfile = $logfile.$$ "
localfiles=`grep "226 Transfer complete" $logfile.$$ | wc -l`
echo " Localfiles = $localfiles"
# convert to numeric format
(( remotefiles = remotefiles ))
# Remove the 1 standard 226 messages from the output
#  (from the mdir command)
(( localfiles = localfiles - 1 ))
print "FTPTAB_RUN:  Remotefiles=$remotefiles  Localfiles=$localfiles"
rc2=0
if  [[ $localfiles -ne $remotefiles ]]
then
    rc2=1
fi
echo "FTPTAB_RUN: FILE COUNT STATUS (rc2)=$rc2"
(( rc = rc1 + rc2 ))
echo "FTPTAB_RUN: EXIT CODE (rc)=$rc"
echo "$rc" > $statusfile
########################################
#Save the sessionlog to logfile
########################################
print "FTBTAB_RUN: Session Log\n"
cat $logfile.$$
#rm $logfile.$$
exit $rc
################################################################################t
```

FTPTAB_PRE.SH

```
############################################################
cd /usr/ftptab/out
compress -v file1.dat
compress -v file2.dat
exit 0
############################################################################
```

Mukesh Dang and Mike Tevlin
System Administrators(USA)

Utility for documenting your system

You should always have a good system back-up (mksysb) of your system, but there are often times when your system is down and you need to quickly obtain information about it. The nodeReport script is designed to record as much information as possible about your system, in text format, for quick reference purposes.

When run, the nodeReport script writes to standard output an indexed list of the commands followed by the output to each command. Ideally this should be run on a weekly or monthly basis, with the output stored on disk, on another system, on tape, on CD-ROM, or even as hardcopy. In this way, all the information will be available quickly when the system is down.

Here is the script:

```
#!/usr/bin/ksh
#
# Script: nodeReport
# Author: Roger Wickings
# Aim:    Display all node details with the aim of documenting the node
#

awk="/usr/bin/awk"
basename="/usr/bin/basename"
cat="/usr/bin/cat"
cut="/usr/bin/cut"
dc="/usr/bin/dc"
df="/usr/bin/df"
domainname="/usr/bin/domainname"
exportfs="/usr/sbin/exportfs"
find="/usr/bin/find"
grep="/usr/bin/grep"
hostid="/usr/bin/hostid"
id="/usr/bin/id"
ifconfig="/usr/sbin/ifconfig"
ls="/usr/bin/ls"
lsattr="/usr/sbin/lsattr"
lscfg="/usr/sbin/lscfg"
lsdev="/usr/sbin/lsdev"
lsfs="/usr/sbin/lsfs"
lslpp="/usr/bin/lslpp"
lslv="/usr/sbin/lslv"
lsps="/usr/sbin/lsps"
```

```
lspv="/usr/sbin/lspv"
lssrc="/usr/bin/lssrc"
lsvg="/usr/sbin/lsvg"
lsvpcfg="/usr/sbin/lsvpcfg"
ps="/usr/bin/ps"
netstat="/usr/bin/netstat"
no="/usr/sbin/no"
rm="/usr/bin/rm"
sed="/usr/bin/sed"
sort="/usr/bin/sort"
tr="/usr/bin/tr"
uptime="/usr/bin/uptime"
uname="/usr/bin/uname"
xargs="/usr/bin/xargs"
ypwhich="/usr/bin/ypwhich"

CRONDIR="/var/spool/cron/crontabs"
TMPDIR="/tmp"

FILELIST="/etc/passwd /etc/group /etc/environment /etc/exports /etc/
hosts /etc/inittab /etc/qconfig /etc/resolv.conf /etc/netsvc.conf"

# functions

createReport()
{
  initialization

  runCommand $hostid
  runCommand $uname -a
  runCommand $uptime
  runCommand $lscfg -v

  # Devices and drivers

  runCommand $lsdev -CH
  LIST=`$lsdev -C | $awk '{print $1}' `
  COMMAND="$lsattr -HE -l"
  runForEach

  runCommand $lslpp -L

  # Network

  runCommand $netstat -rn
  runCommand $netstat -in
  LIST=`$netstat -in | $awk '{print $1}' | $grep -v "Name" | $sort -u `
  COMMAND="$ifconfig"
  runForEach
```

```
  runCommand $no -a
  runCommand $exportfs
  runCommand $domainname
  runCommand $ypwhich

  # Space

  runCommand $df -k
  runCommand $lsfs
  runCommand $lsps -a
  runCommand $lsps -s
  runCommand $lsvpcfg

  runCommand $lsvg
  LIST=`$lsvg | $sort -u `
  COMMAND="$lsvg"
  runForEach
  COMMAND="$lsvg -l"
  runForEach

  runCommand $lspv
  LIST=`$lspv | $grep -v "None" | $awk '{print $1}' | $sort -u `
  COMMAND="$lspv"
  runForEach
  COMMAND="$lspv -l"
  runForEach

  LIST=`$lsvg | $xargs -iX $lsvg -l X | $awk '{print $1}' | $grep -v -e
"LV" -e ":$" | $sort -u `
  COMMAND="$lslv"
  runForEach
  COMMAND="$lslv -l"
  runForEach

  # Processes

  runCommand $lssrc -a
  runCommand $ps -Afl

  # Files

  LIST=`$ls $FILELIST 2>/dev/null `
  COMMAND="$cat"
  runForEach

  # Crontabs

  LIST=`$find $CRONDIR -type f -size +0c -print `
  COMMAND="$cat"
  runForEach
```

```
  outReport

  return
}

initialization()
{
  NODE=`$uname -n `
  USER=`$id -un `

  INDEX="$TMPDIR/$SCRIPT.index.$$"
  REPORT="$TMPDIR/$SCRIPT.report.$$"

  COUNT="Ø"

  echo                                              >> $INDEX
  echo "   Ø.  INDEX ($NODE)"                       >> $INDEX
  echo "   Ø.  INDEX ($NODE)" | $sed "s/./=/g"  >> $INDEX

  return
}

outReport()
{
  $cat $INDEX
  echo
  $cat $REPORT
  echo "\nEnd-of-report."

  $rm -f $INDEX $REPORT

  return
}

runCommand()
{
  command="$*"

  exefile="$1"
  if test ! -x "$exefile"
  then
    return
  fi

  let COUNT=COUNT+1
  number=`echo "$COUNT 1ØØØ + p q" | $dc | $sed "s/^1/ /" | $sed "s/ Ø/
/" | $sed "s/ Ø/  /" `

  echo "$number.  $command"                              >> $INDEX
```

```
  echo                                                    >> $REPORT
  echo "$number.  $command   ($NODE)"                     >> $REPORT
  echo "$number.  $command   ($NODE)" | $sed "s/./=/g"    >> $REPORT
  echo                                                    >> $REPORT
  $command                                                >> $REPORT  2>>
$REPORT

  return
}
runForEach()
{
  for value in $LIST
  do
    runCommand $COMMAND $value
  done
  return
}

# start of main processing

SCRIPT=`$basename $0 `

createReport

exit 0
```

You can see that the function createReport contains all the commands that are run, hence it can easily be amended to include any extra commands that you think are important.

The report produced is in two parts – the index and the main body of the output. I have listed a small portion of both parts below.

For most of our systems the full report is under 0.5MB.

The report index:

```
   0.  INDEX (testbox1)
=======================
   1.  /usr/bin/hostid
   2.  /usr/bin/uname -a
   3.  /usr/bin/uptime
   4.  /usr/sbin/lscfg -v
   5.  /usr/sbin/lsdev -CH
   6.  /usr/sbin/lsattr -HE -l sys0
   7.  /usr/sbin/lsattr -HE -l sysplanar0
```

The main body of the report:

```
   1.  /usr/bin/hostid   (testbox1)
```

```
====================================

0x91f609fb

   2.  /usr/bin/uname -a   (testbox1)
====================================

AIX testbox1 3 4 0043343A4C00

   3.  /usr/bin/uptime   (testbox1)
====================================

  11:11AM  up 95 days,  20:32,  32 users, load average: 0.05, 0.06, 0.05

   4.  /usr/sbin/lscfg -v   (testbox1)
====================================
```

INSTALLED RESOURCE LIST WITH VPD

The following resources are installed on our machine:

- Model architecture: chrp
- Model implementation: multiple processor, PCI bus
- sysplanar0 00-00 system planar
- mem0 00-00 memory
- proc0 00-00 processor.

Roger Wickings
Systems Programmer
FT Interactive Data (UK)

AIX news

IBM has announced Version 2.0 of its WebSphere Host Integration Solution for integrating existing applications with the Web.

It has new and enhanced Web integration capabilities and now includes WebSphere Application Server, WebSphere Studio, and IBM Screen Customizer to exploit legacy data in new applications.

Offered for both intranets as registered users and the Internet for concurrent users with different product content, Version 2.0 is priced per registered or concurrent user.

It's made up of IBM Personal Communications, WebSphere Host On-Demand, WebSphere Host Publisher, WebSphere Studio, Professional Edition, WebSphere Application Server, Advanced Edition, Screen Customizer, and Communications Server for OS/2, AIX, Windows NT, and Windows 2000,

For further information contact your local IBM representative.
URL: http://www.ibm.com/software/webservers.

* * *

Perle Systems has added TruePort for AIX applications across its range of serial servers, which means the company now supports the connection of serial devices to AIX applications via Ethernet-based serial servers.

It allows serial devices connected to a Perle Serial Server to function as if directly connected to the AIX server by mimicking TTY port functionality. Applications written for directly connected serial devices can be deployed across an Ethernet LAN environment without modification.

The software provides transparent access from AIX applications to as many as 500 serial ports on Perle serial servers across a TCP/IP network.

For further information contact:
Perle Systems, 700 Commerce Drive, 5th Floor, Oak Brook, IL 60523, USA.
Tel: (630) 288 4879.
URL: http://www.perle.compress_releases/press_trueport.htm.

* * *

Mercury Interactive has announced Version 7.0 of both its LoadRunner load testing product and WinRunner for functional and regression testing. They have new capabilities, which help determine whether applications will perform as expected and will maintain that performance with increased usage and rapid changes to content and functionality.

Both are available on platforms including AIX, HP-UX, Linux, NT, and Solaris.

The company has added a Web Transaction Breakdown Monitor to LoadRunner, which enables the latter to break down end-to-end transaction response times for the client, network, and server and provide the means to drill-down to the exact page component that may be causing performance slowdowns.

For further information contact:
Mercury Interactive, Building A, 1325 Borregas Ave, Sunnyvale, CA 94089, USA.
Tel: (408) 822 5200.
URL: http://www-svca.mercury interactive.com/products/loadrunner/.