AIX update

69

July 2001

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

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 download a copy of our *Notes for Contributors* from www.xephon.com/contnote.html.

Interesting shell prompts

I often receive comments concerning my interesting-looking shell prompts. “How do I make my prompt look like yours?”, they ask me. Answering this question presents a problem because it has many possible answers. The command required to change a shell prompt depends on the shell. The Bourne shell’s syntax is not the same as the C shell nor the same as Bash’s syntax, etc. In addition, people might want various information to appear in their prompt. Even those happy with a single character prompt might prefer ‘>’ to ‘%’; others may also want the date or time, current working directory, hostname, or a sequence number.

BOURNE SHELL

To change your shell prompt in the Bourne or Korn shells, set the PS1 variable. PS1 is called the ‘primary prompt’ (as opposed to PS2, the secondary prompt). In typical **sh** fashion, you set this variable using the following syntax. Notice how the prompt immediately changes:

```
$ PS1="> "
>
```

I specified a blank after the ‘>’ sign. If you don’t include this, then your commands will look like this:

```
>date
Mon Feb  2 08:45:41 EST 2001
>pwd
/home/werner
```

This result may or may not please you, but these lines look cramped and would certainly annoy me. This kind of change will affect only your current shell (not ‘children’ shells that you might start up from the current shell – unless you subsequently export the PS1 variable like this:

```
> export PS1
```

KORN SHELL

I'll regard the Korn shell as being the most frequently used shell in the AIX world. In addition to the already-mentioned capabilities of the Bourne shell it has some additional prompt enhancements. Its $PWD variable constantly contains the proper working directory. This means that setting PS1 to $PWD puts the working directory name into your prompt.

```
$ PS1='$PWD $ '
/home/werner $
```

Additional environment values are ${LOGNAME}, ${HOSTNAME}, and ${TTY}. They allow the following prompt:

```
$ PS1='
> ${LOGNAME}@${HOSTNAME} on ${TTY}
> [ ${PWD} ] : '

werner@klauser.ch on /dev/pts/4
[ /home/werner ] :
```

If you just want the last component of the directory, use:

```
$ PS1='${PWD##*/} $ '
werner $
```

Maybe even better would be seeing the whole directory name unless you were in one of your own directories. Then you only want to see its path relative to your home directory:

```
$ PS1='${PWD#$HOME/} $ '
/home/werner  $
```

BASH SHELL

The method of changing your prompt in Bash is very similar, but Bash provides a lot of very convenient options if you're interested in adding other information to your prompt. For example, this command:

```
PS1="\u@\h > "
```

puts both the username and hostname in the prompt like this:

```
werner@klauser >
```

Meanwhile, this one:

```
PS1="\h \#> "
```

puts the hostname and a command sequence number in the prompt:

```
klauser 11>
```

To put the current path and sequence number, you would use this syntax:

```
klauser 14> PS1="\w \#> "
/home/werner 15>
```

C SHELL

Changing your prompt using the C Shell is a bit more tricky. The following command changes the prompt each time the **cd** command is issued:

```
alias cd 'cd \!*; set prompt="`pwd`> "'
```

You can replace the **pwd** command with **uname -n** or another command to customize your shell.

MAKING YOUR CHANGES STICK

Where you put your prompt settings so that they become permanent also depends on the shell you are using. Here is a list of shells and their dot files:

- Bourne – .profile
- Korn – .kshrc or .profile
- Bash – .bashrc
- C shell – .cshrc
- Tcsh – .tcshrc.

Werner Klauser
Klauser Informatik (Switzerland)

DNS improvements

The scope of this article is to improve the availablity of the domain name resolution. The DNS system works very well if the referenced nameserver and specially the ‘FIRST=primary nameserver’ is available as an IP node. Then the request to resolve an IP alias to an IP address is sent to the secondary nameserver if the primary DNS process is not responding. But what is going on if the primary nameserver had a crash and is not available on the IP network, or is available but through a network problem it is not reachable any more? In this situation you cannot *ping* the primary domain nameserver.

In this case, after a period of time, the internal IP timeout comes up and signals the upper layer that the DNS server node is not reachable under the requested IP address. Then the secondary nameserver will be asked instead.

The delay we are speaking about is approximately 1 minute and 15 seconds. This is the time the user has to wait until the requested hostname will be resolved.

In most situations this occurs when we do maintenance or when a system crashes. Then the phone does not stop ringing, because the user community is complaining about poor system performance, from their point of view. They are confused, and some batch jobs on the server which communicate to others and use the DNS services are confused too. Process queues fill up with ones waiting for real-name resolution, and time-critical applications time out.

SOLUTION

In this situation the following script (and keeping a cool head) will help. The script is activated every five minutes through the crontab and checks whether the DNS Server, which is currently the first, is available and whether this is the best solution for the server. If not the script does some adjustments to minimize the name resolution time and the administration time too.

PREREQUISITES

The script is a simple Korn shell source:

1 The system where it is running must be a DNS client using */etc/resolv.conf*.

2 The script has been tested with AIX 4.2.1, 4.3.2, and 4.3.3.

INSTALLATION INSTRUCTIONS

With root permission, **cd** in the */etc* directory and copy the *resolv.conf* file to *resolv.conf.1* and *resolv.conf.2*. Then adjust the two files so that the *.1* file has the real primary DNS as its first nameserver in the list and the *.2* file has the secondary DNS as its first. You can have more than two best-solution *resolv.conf* files. If you do, you have to modify the script. Put the script in a directory of your choice, for instance name the script *DNS_best_srv*, and do a **chmod u+rx** and a **chmod rw-go**.

Put the following entry in the crontab of the root user:

```
0,5,10,15,20,25,30,35,40,45,50,55 * * * * /location_of_script/
DNS_best_srv >> /tmp/console 2>&1
```

DNS_BEST_SRV

```
#!/usr/bin/ksh
#
# Name : DNS_best_srv
#
# Function : check the current DNS resolution - activate
#            a better one - if available
#
# author : imhotep/2001-03-22
#
# ----------------------------------+
# define  useful function           !
# ---------------------------------- ----------------------------------+
#                                                                      !
# check_if_dns_works ... return code >0                                !
# if the first nameserver entry in file specified in the first         !
# parameter replies to an SOA request of an domain - where we know     !
# that this domain exists - with an answer - like                      !
#     refused                        or                                  !
#     No response                    or                                  !
#     Connection timed out           or                                  !
#     error                          or                                  !
#                                                                        !
# if we get another answer - we return Ø to the calling program          !
# -----------------------------------------------------------------------+
check_if_dns_works ()
{
integer ixx
#
# gather the first nameserver entry -
#
cat $1 | grep nameserver |  read skipname dns_ip
#
domain_which_must_exist=xxxxxxx   ## <<--- Adjust with your domain
{
  echo "server $dns_ip"
  echo "ls -t SOA ${domain_which_must_exist} "
  echo "exit "
}  | nslookup 2>&1 \
 | egrep '(refused|No response|Connection timed out|error)' \
 | wc -l | read ixx
#
return $ixx
}
#
# ---------------------------+
# Main processing            !
# ---------------------------+
#
origf=/etc/resolv.conf
echo $Ø `date` start
#
# check if this is a DNS client
#
if test ! -f ${origf}
then
  echo $Ø this is not a DNS client - script only works
  echo $Ø on DNS clients
  exit 99
fi
#
# Loop through the best solutions in descending order
# This is central to the process that the .1 file contents are
# the best solution for DNS resolution and .2 file contents are the
# second best solution - and all following .x+1 files may be solutions
# but not so good as the preceding .x file
```

```
#
for dnsprio in 1 2  # <<-- modify your solutions
do
resfile=/etc/resolv.conf.${dnsprio}
if test -f ${resfile}
then
#
# when a better solution file exists - we can start compare
# and testing
  diff -i ${origf} ${resfile} > /dev/null 2>&1
  if test ${?} -eq 0
  then
#
# The actual resolv.conf file and the .x file are equal
# now let's check if DNS still works
#
    check_if_dns_works ${resfile}
    if test ${?} -eq 0
    then
#
#    The actual DNS resolution works well - so we have to do nothing
#
        exit
    else
#
#    The actual DNS resolution doesn't work well - so loop to the
#    the next offered solution to check
        :
    fi
   else
#
# The resolv files are not equal - they differ
#
    check_if_dns_works ${resfile}
    if test ${?} -eq 0
    then
#
#   That's our chance - the old one is not the best - or doesn't
#   work well
      cp ${resfile} ${origf}
      if test ${?} -eq 0
      then
        echo $0 DNS Resolving changed form ${origf} to ${resfile}
        exit
      else
        echo $0 Error writing new DNS resolution file error ${?}
        echo can not switch from ${origf} to ${resfile}
        exit
      fi
    fi
  fi
fi
done
echo $0 `date` end
```

SAMPLE /ETC/RESOLV.CONF FILE

```
cat resolv.conf.1
nameserver      194.50.145.3
nameserver      194.50.145.4
domain  xxxxx

# and the second - best one

cat resolv.conf.2
nameserver      194.50.145.4
nameserver      194.50.145.3
domain  xxxxx
```

Imhotep
Unix System Administrator (Austria)

Improving I/O performance using LVM striping

Other than installing faster disk drives on the system, the best way to improve the I/O performance to disk is to spread the I/O workload across as many physical disks as possible. You can simply expand a filesystem across two disks, but this just splits the filesystem in half. This may improve performance to a degree when reading random data that is physically located on different disks, but all data within the same half of the filesystem will still use the same physical disk.

Disk ‘striping’ physically staggers small units of data across one or more physical disks, allowing those disks to more-evenly share the workload of reading and writing data. For logical volumes containing randomly-accessed data, such as filesystems, making use of multiple concurrent disks can dramatically improve performance.

STRIPING AT THE PHYSICAL PARTITION LEVEL

In AIX 4.1 and earlier, you can implement your own striping mechanism by using a physical partition ‘map’ to create a logical volume. In doing so, you can indicate which physical partitions, in which order, and on which disks, should be used to make up the logical volume. A simple physical partition map, which contains the physical volume name and the physical partition number on the disk, looks something like this:

```
hdisk1:75
hdisk2:100
hdisk1:76
hdisk2:101
```

The logical volume may then be created using the command:

```
    mklv -y newlvname -m /tmp/lvmap vgname 4
```

In this example, the logical volume *newlvname* will be created in the *vgname* volume group. The number of partitions (4) must match the number of entries in the *lvmap* file, and the physical volumes specified in the *lvmap* file must exist in the specified volume group.

In AIX 4.1, IBM made things a little easier by automatically staggering the physical partitions across disks when the user created a logical volume using a maximum Inter-PV policy. With this policy, a logical volume could be created without the use of a map by using the command:

```
mklv -y newlvname -ex -ac vgname 4 hdisk1 hdisk2
```

The **–ex** argument indicates the partitions should be staggered across the disks. Because LVM will determine the default physical partitions to use, we add the **–ac** argument (*Intra-PV policy*) to specify that we want the logical volumes to use available partitions nearest to the centre region of the disks.

Existing logical volumes may also be re-mapped by changing the Inter-PV policy to 'maximum', then using the *reorgvg* command to reorganize the logical volumes on disk:

```
chlv -ex -ac -u2 mylvname
reorgvg vgname mylvname
```

This command will reorganize the partitions in the *mylvname* logical volume to use available partitions nearest the centre region of up to two disks (-u2) in the *vgname* volume group. You may specify multiple LV names with the **reorgvg** command, and all of the specified LVs will be reorganized, giving priority for physical partitions to those specified first in the list.

In most cases, this form of physical partition striping improves performance, but still does not provide the most efficient distribution of the I/O workload. This is because a physical partition was typically four megabytes (determined when the volume group is created), so any data read within the same physical partition would still utilize the same disk.

LOGICAL VOLUME STRIPING

With AIX 4.2, a new LVM option for striping data within a single physical partition was introduced. LVM striping breaks down each physical partition into very small units, and stripes these units across as many physical devices as desired. The stripe size may be between 4KB and 128KB in size. The smaller the stripe size, the more the data is fragmented across the disks (a good thing), and the more likely that a single read or write operation will utilize multiple physical drives.

LVM striping may only be implemented when a logical volume is first created. To stripe a logical volume, use the **–S** flag to the **mklv** command as in the following example:

```
mklv -y newlvname -S 16K vgname 4 hdisk1 hdisk2
```

With this command, the logical volume, *newlvname*, will be created in the *vgname* volume group and striped across physical volumes *hdisk1* and *hdisk2*. Note that the number of physical partitions is four and stripe size is set to 16KB.

The stripe width is determined by the number of disks you specify with the mklv command. If you do not specify any physical volume names, then you must indicate the stripe width using the **–u** (upperbound) flag. If the upperbound flag is used, LVM will stripe the logical volume across only that number of physical volumes. The drawback is that you cannot specify which physical volumes will get used.

Note also that the stripe width must be a factor of the number of physical partitions in the logical volume. For example, if the LV uses ten physical partitions, you cannot stripe across three PVs since the LV must be distributed evenly across the disks. In this case, you must either use 12 partitions or use only two physical volumes.

SEQUENTIAL VERSUS RANDOM I/O

The important thing to understand about striping is that it improves I/O performance when randomly accessing data within a logical volume. Filesystem data is typically fragmented within a logical volume, so concurrently accessing many files within a filesystem will benefit from striping. However, when reading data within a logical volume sequentially, or when reading a large file sequentially, you will not be utilizing multiple disks concurrently (you will utilize multiple disks, just not at the same time).

When reading data sequentially, there may be some performance benefit when a single read requires data from different stripes. For example, if issuing a single read operation for 128KB to a logical volume with a 32KB stripe size, two stripes will be read from the first disk and two stripes from the second.

WHAT TO STRIPE

As stated above, logical volumes that benefit most from striping are those that are accessed randomly, particularly those containing filesystems. Filesystems containing data files are good candidates, while filesystems containing application programs may not be since they are accessed less frequently, and programs used repeatedly are more likely to be found in cache than on the disk.

Logical volumes you should not stripe include JFS logs, because they are only read and written sequentially and in small increments. Paging space should also not be striped. Since the system will automatically spread the allocated memory across multiple paging spaces (its own built-in striping, so to speak), it's better to create an additional paging space on a second disk than to stripe a single paging space.

IIMPACT ON AVAILABILITY

By implementing striping, you have increased the *single-points-of-failure* that may cause the data to become unavailable. In a non-striped environment in which a filesystem is solely contained on one disk, only the failure of that disk will cause unavailability of the data. However, if the data is striped across two disks, the failure of either of the disks will cause the data to be unavailable. Likewise, if you stripe across three disks, the single points of failure increase to three, etc. It is therefore wise to limit which logical volumes you choose to stripe (based on whether I/O performance is currently an issue), and the number of disks you choose to stripe across.

WHAT STRIPE SIZE TO USE

Whether reading or writing sequentially or randomly, each I/O request must be translated into physical requests to the disk device(s). If you are issuing a large read request to a striped logical volume with a very small stripe size, this will result in many separate physical I/O requests, possibly losing much of the performance gained by striping. If you are reading from a filesystem with many small files, a smaller stripe size may be of benefit, since each I/O request is for a small amount of data. A good rule-of-thumb is to use a stripe size no smaller than half the typical size of a read or write request.

Anthony Johnson
AIX System Back-up Software Developer
Storix Software (USA)

Tidy up before you go!

Do any of your systems suffer from 'multi-administrator' syndrome – where a system has been 'administered' by several people over a period of time who each failed to follow documented procedures, tried to out-do other personnel, or were just lazy and didn't tidy up? Well, this article might just be of interest to you.

All too often machines are installed by an administrator copying programs and data with the installer, really meaning to tidy up those unnecessary empty directories, or delete those testing IDs, but never quite getting round to it. The 'mess' left is normally harmless: a directory which was NFS exported and now does not exist will cause an error message on start up but nothing more. However, a bad entry in *inittab* or *crontab* can cause problems on reboot, or 'no-boot', as you may find.

There is plenty of help with checking a variety of parts of an AIX system. IBM provides some standard Unix (and AIX-specific) programs to help with checking the user database and associated files. *AIX Update* has given you plenty to help you along and the following program (providing you have Perl installed) will add an extra string to your 'admin bow'.

IBM-SUPPLIED PROGRAMS

Below is a non-exhaustive list of the supplied programs to help check the validity of your (AIX) system. Most end with 'ck', so to find a potentially longer list, *cd* to */usr/sbin* and *ls *ck*. Then read the **man** pages for the commands listed. The programs are:

- pwdck (or *pwck* on Sun, HP, and Tru-64 Unix) checks the validity of (and with the appropriate flag, corrects) the password information in the password files. Note that *pwdck* does not check NIS password data.
- grpck (also on Sun, HP, and Tru-64 Unix) checks the validity of all the members of the groups listed in the local group file (*/etc/group*). Again, no NIS information is checked.

- usrck (not on other Unix, though HP does have *authck*) checks that all users listed in the */etc/passwd* file have sensible values in the relevant files in the */etc/security* directory.
- tcbck audits the trusted computing base installation using the definitions in the */etc/security/sysck.cfg* file.
- sysck (not on other Unix) is part of the **installp** process really. However, you can use it to check the integrity of the files installed with **installp**, which have got definitions in the */etc/security/sysck.cfg* file.

There are others, but I have just highlighted a few.

Other commands do a little checking at the time of alteration. For example **cron** does do some field validation on altering a line and will throw it out if entered incorrectly. However, it only goes so far. The command entered into crontab is not checked and, if not spotted, can ultimately fill the */var* filesystem with mail messages. This is where the following program comes in.

The program itself is relatively simple and is not exhaustive. It deals with three areas of an AIX system (although it can be modified to check any Unix system without too much editing). The three areas are inittab, crontab, and NFS/filesystems.

The inittab file is not checked for correctness until it is accessed by the *init* process (usually at system start up). If you are a good housekeeper, then you will use the *chitab*, *mkitab*, and *lsitab* commands to help with altering the file. However, at a guess, you will be like the rest of us and just *vi* it. The program below does not check the validity of each entry, only the validity of each command which *init* will try to execute for each entry. It tries to resolve continued lines (see **man** pages) and will check for the execution status of each command. Errors are produced for commands that are directories or just not executable. Warnings (if the **–w** option is used) are produced if the command is a symlink or is not specified using its full path.

The crontab file is already checked for format validity. So once again the program attempts to check only that the command specified for each line is executable, and the same conditions are flagged as for inittab. All crontabs are checked in the */var/adm/cron/crontabs* directory; however, as you can specify any file as input to **cron** for your crontab, the program accepts the **–c** option with a filename and the commands in that file will be checked.

The NFS information checked is in two parts. Firstly, any filesystems that are exports through the */etc/exports* file are checked for validity (or you can use the **–n** option and specify an alternative export file). For each exported directory, the 'root' and 'access' parts are checked for hostname validity in either a DNS or non-DNS (*/etc/hosts*) environment. Errors are produced, should a directory or a hostname not exist.

The second part of NFS checking is a by-product of checking the validity of the */etc/filesystems* file. For local filesystems, the logical device file is checked as well as the mount point. For remote filesystems, an attempt is made to mount the filesystem to check its validity (after checking that the host exists and can be *ping*ed). Again error messages are displayed for severe errors. However, warnings (**–w**) are produced should the remote host fail to be *ping*ed.

(As a by product, the program checks the validity of the $PATH environment variable for the 'root' user-id.)

THE PROGRAM

```
#!/usr/bin/perl

$|=1;

use strict;
use File::Basename;
use Getopt::Std;
use vars qw/ $opt_n $opt_c $opt_w /;

my @pathdirs=split(":",$ENV{"PATH"});
my
($sec,$min,$hour,$mday,$mon,$year,$wday,$yday,$isdst)=localtime(time);
$mday=sprintf("%02d",$mday);
my @monthname=("Jan", "Feb", "Mar", "Apr", "May", "Jun", "Jul", "Aug",
"Sep", "Oct", "Nov", "Dec");
```

```
$mon=$monthname[$mon];
$year+=1900;
my $rundate=join("-",$mday,$mon,$year);

sub process_symlink {

  # Replace first "../" with real directory name if required

    my $link=shift;
    my $dir;
    my @dir;

    @dir=split(/\//,dirname($link));
    pop(@dir);
    $dir=join("/",@dir);
    $link=readlink $link;
    if ( grep /^\.\./, $link ) {
      $link=~s/\.\.\///;
      $link=$dir."/".$link;
    }
    return $link;
}

sub file_status {

  # Check status of a command name ie exists/directory/symlink etc

    my $command=shift;
    my $dir;
    my $filenm;
    my $return_code=0;

    if ( ! -e $command ) {
      $return_code=4 if ($opt_w);
      FORLOOP:
      foreach $dir (@pathdirs) {
        opendir(DH,$dir) or print "$rundate : WARNING : cannot open
directory $dir in environment PATH\n";
        while(defined($filenm=readdir(DH))) {
          next if ( ! -f "$dir/$filenm" );
          if ( $command eq $filenm ) {
            $command=$dir."/".$command if ( $command eq $filenm );
            last FORLOOP;
          }
        }
      }
    }
    while ( -l $command ) {
      $command=process_symlink($command);
      $return_code=3 if ($opt_w);
    }
    if ( -d $command ) {
      $return_code=2;
    }
    if ( ! -x $command ) {
      $return_code=1;
    }
  return $return_code;
}

sub check_host {

  # Check to see if a hostname exists in /etc/hosts or via DNS

  my $to_check=shift;
  chomp $to_check;

  if ( -e "/etc/resolv.conf" ) {
    open(CH,"nslookup $to_check 2>&1|");
    while(<CH>) {
      if (/Non-existent/) {
        close(CH);
        return 1;
      }
    }
    close(CH);
    return 0;
  } else {
    open(HF,"/etc/hosts");
    while (<HF>) {
      if ( grep /$to_check/i, $_ ) {
        close(HF);
        return 0;
      }
    }
    close(HF);
  }
  return 1;
}

#
#   Main code starts here
#
if ($< != 0) {
   print "ERROR : Need to run as root user.\n";
   exit 1;
```

```
}
getopts("n:c:w");
       # Allow -n (alt exports file) -c (alt crontab file) -w (warnings)
my @command;
my $inline;
my $hostname;
my @line;
my @lines;
my @to_process;
my $file;
my @crontabs;
my $filesystem;
my $host_count=0;
my $count=0;
my $remotefs=0;
my $failed=0;
my $rc;
my $nfsexport_file="/etc/exports";
$nfsexport_file=$opt_n if ( $opt_n ne "" );
if ( $opt_c ne "" ) {
  if ( -e $opt_c ) {
    push(@crontabs,$opt_c);
  } else {
    print "$rundate : ERROR : Command line crontab file $opt_c does not
exist\n";
  }
}

print "$rundate : Checking INITTAB\n";
print "$rundate : ================\n";
open(IT,"/etc/inittab") or die "$rundate : ERROR : Cannot open /etc/
inittab\n";
while($inline=<IT>) {
  chomp $inline;
  next if (grep /^:/, $inline);
                                  # Ignore comment lines in inittab
  if ($inline =~ s/\\$//) {
                                  # Make continuation lines one
    $inline .= <IT>;
    redo unless eof(IT);
  }
  @line=split(/:/,$inline);
  next if $line[2] eq "initdefault";   # Ignore init line
  @command=split(/ /,$line[3]);
  chomp $command[0];
  $count++;
  if ( ($rc=file_status($command[0])) != 0 ) {
    $failed++;
    print "$rundate : ERROR : Command '$command[0]' is a DIRECTORY\n" if
( $rc == 2 );
    print "$rundate : ERROR : Command '$command[0]' is NOT executable\n"
if ( $rc == 1);
    print "$rundate : WARNING : command '$command[0]' is a SYMLINK\n" if
( $rc == 3);
    print "$rundate : WARNING : command '$command[0]' does not have a
full path\n" if ( $rc == 4);
  }
}
close(IT);
print "$rundate : Checked $count commands in inittab, $failed errors\n";
print "$rundate : ------------------------------------------------------
-----\n";

print "$rundate : Checking CRONTABs\n";
print "$rundate : =================\n";
opendir(DH,"/usr/spool/cron/crontabs") or die "$rundate : ERROR : Cannot
open the crontabs directory\n";
while(defined($file=readdir(DH))) {
  next if ! -f "/usr/spool/cron/crontabs/$file";
  push(@crontabs,"/usr/spool/cron/crontabs/$file");
}
closedir(DH);
foreach $file (@crontabs) {
  $count=0;
  $failed=0;
  open(CF,$file);
  while(<CF>) {
    next if (/^\$*#/);
    $_ =~ s/^\s+//;
    next if (/^$/);
    $count++;
    @command=split(/\s+/,$_);
    chomp $command[5];
    $rc=file_status($command[5]);
    if ( ($rc=file_status($command[5])) != 0 ) {
      $failed++;
      print "$rundate : ERROR : Command '$command[5]' is a DIRECTORY\n"
if ( $rc == 2 );
      print "$rundate : ERROR : Command '$command[5]' is NOT
executable\n" if ( $rc == 1);
      print "$rundate : WARNING : command '$command[5]' is a SYMLINK\n"
if ( $rc == 3);
      print "$rundate : WARNING : command '$command[5]' does not have a
full path\n" if ( $rc == 4);
    }
  }
  close(CF);
  print "$rundate : Checked $count commands in $file, $failed errors\n";
}
print "$rundate : ------------------------------------------------------
-----\n";

$count=0;
$failed=0;
print "$rundate : Checking NFS exports\n";
print "$rundate : ====================\n";
if ( -e $nfsexport_file ) {
  open(XH,$nfsexport_file);
  while(<XH>) {
    next if (/^#/);                   # Ignore comment lines in exportfs
    @line=split(/ /,$_);
    chomp $line[0];
    $count++;
    if ( file_status($line[0]) != 2 ) {
      $failed++;
      print "$rundate : ERROR : Directory $line[0] is in error for
NFS\n";
    }
    $file=shift(@line);
    @line=split(/\,/, $line[0]);
    while ( scalar(@line) != 0 ) {
      $_=shift(@line);
      SELECT: {
        /^access|^-access/ and do {
          @to_process=split(/\:/,(split(/\=/,$_))[1]);
          while ( scalar(@to_process) != 0 ) {
            $host_count++;
            if ( check_host($to_process[0]) ) {
              chomp $to_process[0];
              print "$rundate : ERROR : Host $to_process[0] does not
exist for 'access' for export $file\n";
              $failed++;
            }
            shift(@to_process);
          }
          last;
        };
        /^root|^-root/ and do {
          @to_process=split(/\:/,(split(/\=/,$_))[1]);
          while ( scalar(@to_process) != 0 ) {
            $host_count++;
            if ( check_host($to_process[0]) ) {
              chomp $to_process[0];
              print "$rundate : ERROR : Host $to_process[0] does not
exist for 'root' for export $file\n";
              $failed++;
            }
            shift(@to_process);
          }
          last;
        };
      }
    }
  }
  close(XH);
  print "$rundate : Checked $host_count hosts for $count directories in
$nfsexport_file, $failed errors\n";
} else {
  print "$rundate : ERROR : $nfsexport_file file does not exist\n";
}
print "$rundate : -----------------------------------------------------
-----\n";

@lines=();
@to_process=();
$count=0;
$failed=0;
print "$rundate : Checking Filesystems files\n";
print "$rundate : ========================\n";
open(FH,"/etc/filesystems") or die "$rundate : ERROR : Cannot open /etc/
filesystems file\n";
while(<FH>) {
  next if (/^\$*\*/);           # Ignore comment lines in filesystems
  $_ =~ s/^\s+//;
  next if (/^$/);               # Ignore blank lines too
  push(@lines,$_);
}
close(FH);
while (@lines) {
  @to_process=();
  do {
    $inline=pop(@lines);
    if ( grep /nodename/, (split /=/, $inline)[0] ) {
      chomp $inline;
      @line=split(/\=/,$inline);
      $line[1] =~ s/^\s+//;
      $hostname=$line[1];
    }
    if ( grep /dev/, (split /=/, $inline)[0] ) {
      chomp $inline;
      @line=split(/\=/,$inline);
      $line[1] =~ s/^\s+//;
      $filesystem=$line[1];
    }
  } until ($inline =~ m/\/[A-Z,a-z,0-9,_,-]*\:/);
  $count++;
  chomp $inline;
  chop $inline;
                           # Remove trailing : from filesystem name
  if ( -d $inline ) {
    if ( $hostname ) {
      $remotefs++;
      if ( check_host($hostname) ) {
        $failed++;
        print "$rundate : ERROR : Invalid hostname $hostname specified
for filesystem $inline\n";
      } else {
        $rc=0;
        open(PH,"/usr/sbin/ping -c1 -q $hostname 2>&1|");
        while(<PH>) {
          $rc=1 if (/100% packet loss/);
        }
        close(PH);
        if ( $rc ) {
          if ( $opt_w ) {
            $failed++;
            print "$rundate : WARNING : Could not ping host $hostname to
check filesystem $filesystem\n";
          }
        } else {
          mkdir("/tmp/$$",0777) or die "$rundate : ERROR Cannot create
temporary directory in /tmp";
          open(CH,"/usr/sbin/mount $hostname:$filesystem /tmp/$$
2>&1|");
          while(<CH>) {
            if (/1831-011/) {
              $failed++;
              print "$rundate : ERROR : unable to mount $filesystem from
host $hostname due to access permissions\n";
            }
            if (/1831-019/) {
              $failed++;
              print "$rundate : ERROR : filesystem $filesystem does not
exist on host $hostname\n";
            }
          }
          close(CH);
        }
        system("/usr/sbin/umount /tmp/$$ > /dev/null 2>&1");
        rmdir("/tmp/$$");
      }
      $filesystem="";
    }
    if ( $filesystem ) {
      if ( ! -e $filesystem ) {
        $failed++;
        print "$rundate : ERROR : Invalid device $filesystem specified
for filesystem $inline\n";
      }
    }
  } else {
    $failed++;
    print "$rundate : ERROR : Filesystem $inline does not exist\n";
  }
  $hostname="";
  $filesystem="";
}
print"$rundate : Processed $count filesystems ($remotefs remote),
$failed errors.\n";
print"$rundate : -------------------------------------------------------
----\n";
```

As I stated earlier, the program is not exhaustive. It can easily be extended to include such things as checking that the hosts in the */etc/qconfig* file for remote printers exist. I leave it up to you.

Phil Pollard
Unix Systems Administrator (UK)

Implementing I/O multi-pathing using AutoPath – part 2

This month we conclude the article looking at I/O multi-pathing using AutoPath.

The datapath query device command displays information about a single or all available devices.

Syntax:

```
    datapath query device [device number]
```

If a device number is absent, the command displays information about all available devices.

For example:

```
datapath query device

Total Devices : 2


Dev#:   Ø     Device Name: vpathØ
=================================
Type     Serial    Path#    State     Mode      Select     Errors
OPEN-8   1 Ø336      Ø      OPEN      NORMAL     2Ø122       Ø

Dev#:   1     Device Name: vpath1
=================================
Type     Serial     Path#   State     Mode      Select     Errors
OPEN-8   1 Ø342      Ø      OPEN      NORMAL     415472      Ø
                     1      OPEN      NORMAL     415472      Ø
```

The fields are:

- Dev # – device number.
- Device name – device name.
- Serial – the logical unit (LUN) serial number for the device.
- Path # – path number.

- Type – device product ID from inquiry data. In our case OPEN-8*2 14GB disk.
- Operational state of the device:
  - OPEN – the path is in use.
  - CLOSE – the path is not in use.
  - DEAD – the path is no longer being used. It was either removed by Auto Path XP because of errors or removed manually by means of **datapath set device** command.
- Mode – either NORMAL or OFFLINE.
- Select – the number of times the device was selected for input or output.
- Errors – the number of errors on a path connected to the device.

The datapath set adapter command sets all paths attached to an adapter to either ONLINE or OFFLINE. It is used to recover a path after a malfunction is repaired.

Syntax:

```
    datapath set adapter   [adapter number] [online | offline]
```

Note: this command will not remove the last path to a device.

For example:

```
    datapath set adapter Ø online
```

The datapath set device command sets the path of a device to either ONLINE or OFFLINE. It is used to recover a path after a malfunction is repaired.

Syntax:

```
datapath set device [adapter number] path path number [online | offline]
```

Note: in order to prevent data access failure, this command will not let you remove the last path to the device.

For example:

```
datapath set device 0 path 0 online
```

The mkvg4vp command creates a volume group using vpath devices. This command is equivalent to the AIX command **mkvg** that is operating on hdisk devices.

Syntax:

```
mkvg4vp [options] <volume group name> <vpath0> [ <vpath1> …]
```

Parameters:

- options – identical to options in the **mkvg** command.
- volume group name – the name of the volume group to be created.
- vpath0, vpath1 .. – names of the vpath devices to add to the created volume group.

The hd2vp command converts a volume group that is using hdisk devices to use vpath devices instead. All filesystems residing on the processed volume group must be unmounted during the execution of this command.

Syntax:

```
hd2vp <volume group name>
```

where volume group name is the name of the volume group to be converted to use Auto Path XP-based vpath devices.

The hd2vp command converts a volume group that is using vpath devices to use hdisk devices instead. All filesystems residing on the processed volume group must be unmounted during the execution of this command.

Syntax:

```
vp2hd <volume group name>
```

where volume group name is the name of the volume group to be converted to use hdisk devices.

The lsvpcfg command shows the relationship between vpaths and hdisk devices.

Syntax:

```
lsvpcfg
```

Lsvpcfg needs no parameters.

Output fields:

- Field 1 shows the vpath number.
- Field 2 shows the status of the device and the volume group name if the volume group has been converted to use Auto Path XP vpath devices. The **pv** flag indicates which hdisk device is configured as the physical volume within the volume group.
- Field 3 shows the serial number of the disk array and the LDev number of the internal disk.
- Field 4 shows the configured hdisks for the virtual path device. It also shows the status of the disk. If the volume group has not been converted yet, the volume group name is shown here as well. The **pv** flag indicates which hdisk device is configured as the physical volume within the volume group.

The extendvg4vp command adds a vpath device to an existing volume group. This command is the equivalent AIX command **extendvg** when using hdisk devices. The PVID must be defined to the vpath device before executing the extendvg4vp command.

Syntax:

```
extendvg4vp [- f] < volume group name > < vpath1 > < vpath2 > ….
```

Parameters:

- -f – forces the vpath to be added to the specified volume group unless it is a member of another volume group in the Device Configuration Database or of a volume group that is active.
- volume group name – the volume group one wants to add vpath devices to.

- vpathX – the vpath device one want to add to a volume group.

The dpovgfix command updates the Auto Path XP-related ODM entries and maps the PVID of vpaths in a volume group correctly.

Syntax:

```
    dpovgfix < volume group name >
```

where volume group name is the volume group one wants to update the ODM entries for.

For example, when the PVID of **datavg** is defined to vpath0, hdisk1 and hdisk2, use this command. Then PVID is defined to vpath0 only.

The **pv** flag indicates which hdisk device is configured as the physical volume within the volume group:

```
# lspvcfg
…
vpath0 (Avail pv datavg) 00030023_ 0009__ =hdisk1 (Avail pv vgora1)
hdisk2
(Avail pv vgora1)
# dpovgfix datavg
# lspvcfg
…
vpath0 (Avail pv datavg) 00030023_ 0009__ =hdisk1 (Avail) hdisk2 (Avail)
```

The xpinfo command provides a mapping between the device file(s) on AIX and the associated port and LDev on the SureStore E Disk Array XP256 or XP512. By default, this utility will provide a name cross-reference of all LUNS connected to the array.

Syntax:

```
    xpinfo -[au] [-v] -[l|p] -[i|c|h|r|d[char]] [-f raw_device]
```

Parameters:

- -a – attempt to report information even for devices that are not ‘OPEN’ type devices. For such devices only the target ID, LUN ID, product ID (type), device size, and firmware revision will be reported.

- -c – displays Continuous Access and Business Copy volume designation information. Data fields include subsystem number, CT group number, CA volume designation, and BC volume designation for each of the three MU numbers.
- -d[char] – displays a single line of data per logical device. Each data field is delimited by a comma or by ‘char’ if specified. The ordering of the data fields is as follows:
  - device_file, target_id, LUN_id, port_id, CU:LDev, type, device_size, serial#, code_rev, subsystem, CT_group, CA_vol, BC0_vol, BC1_vol, BC2_vol, ACP_pair, RAID_level, RAID_group, disk1, disk2, disk3, disk4.
- -f raw_device – report information only for the specified logical device. By default (-f not specified), this tool will report on all ‘OPEN’ type devices.
- -h – displays the help message.
- -i – displays identification information. Data fields include target ID, LUN ID, port ID, LDev number, LUN type (product ID), device size, serial number, and firmware revision.
- -l – sort output by LDev number.
- -p – sort output by port number.
- -r – displays RAID configuration information. Data fields include ACP pair number, RAID level, RAID group, and disk mechanisms.
- -u – create report using updated **ioscan** list. By default, the report is created using the kernel image of the last **ioscan**.
- -v – display **xpinfo** version. The **-v** option has precedence over all other options. If the **-v** option is specified along with other output options, only the **xpinfo** revision is displayed.

  By default (options i, c, r, and d not specified), **xpinfo** will display in verbose format.

For example, the following command will provide a mapping of the disk device /dev/hdisk66:

```
       xpinfo -f /dev/hdisk66


Device File : /dev/hdisk66                              Model : XP256
       ALPA : ef                                     Serial # : 00035791
     Target : 00                                    Subsystem : 7700
        LUN : 02                                     CT Group : ---
       Port : CL1J                                  CA Volume : SMPL
    CU:LDev : 00:0f                               BC0 (MU#0) : SMPL
       Type : OPEN-8*2                             BC1 (MU#1) : SMPL
       Size : 14014 MB                             BC2 (MU#2) : SMPL
   Code Rev : 5247                                 RAID Level : RAID5
 Disk Mechs : R100   R110   R120   R130            RAID Group : 1-2
   Port WWN : 50000e100000537a                       ACP Pair : 1
```

This is the verbose format output, which includes all data gathered for the device.

TROUBLESHOOTING AND DIAGNOSTICS

When a controller malfunctions, the failover action automatically executes. After the path is repaired, use the following procedure to recover the path:

1 Use the **datapath set adapter** command to change the status of a path from OFFLINE to ONLINE. You must execute this command immediately after the recover operation.

2 Confirm the state of the adapter by using the **datapath query adapter** command**.**

The Auto Path XP software logs error conditions into the AIX errlog system. Included here are the DPO error log messages. To check whether the Auto Path XP generated an errlog message, enter this command:

```
errpt - a | grep VPATH
```

Errors:

- VPATH_ XBUF_ NOMEM – allocate kernel-pinned memory failed. When trying to open Auto Path XP vpath file, it tries to allocate memory for later I/O transfer. If the system returns kernel-pinned a null pointer to the caller, kernel- pinned memory is not available. The device File Open fails.

- VPATH_ DEVICE_OPEN – the Auto Path XP device file fails to open one of its paths (hdisks). A vpath device open is successful if at least one attached path opens. If all vpath device paths fail to open, then the vpath device open fails.
- VPATH_ DEVICE_OFFLINE – a path of the Auto Path XP vpath fails the I/O request after several attempts at retrying, and its path status is set to DD_ PATH_DEAD. This path is taken offline. Use the **datapath** console command program to set the offline path to online.

The Auto Path XP supports AIX trace functions. The traces IDs for the software are **0x2F7** and **0x2F8**. Trace ID **0x2F7** traces the full path of the algorithm. Trace ID **0x2F8** traces routine entry, exit, and error paths of the algorithm. To use, manually turn on the trace function before the program starts to run, then turn off the trace function either after the program stops, or any time you need to read the trace report. To start the trace function, enter:

```
trace -j 2F7,2F8 -o /tmp/trc.out -a
```

To stop the trace function, enter:

```
trcstop
```

To read the report, enter:

```
trcrpt /tmp/trc.out | more
```

To perform the AIX trace function, you must have the **bos.sysmgt.trace** fileset installed on your system. The output of the trace must be located in a directory with large amount of free disk space.

REFERENCES

The following are useful references:

- *HP SureStore E AutoPath XP, Release Notes Version 1.00.01*.

- *Auto Path XP, Version 1 Revision 1*, SC26-7291-00.
- *Implementing Fibre Channel Attachment on the ESS*, SG24-6113-00.

Alex Polyak
System Engineer
APS (Israel)

Automated Informix database back-ups

This article describes how to automate unattended Informix database back-ups on an AIX system. Normally the Informix utility **ontape** is used to back-up the databases to tape, and the desired level is specified according to company policy. That is, a level-0 back-up does a full database back-up, a level-1 incremental back-up backs up all changes since the last level-0, and a level-2 backs up all changes since a level-1.

It is common for a level-1 or level-2 back-up to fit on one tape, but a level-0 may take multiple tapes, depending on the tape capacity and database sizes. This is where the inconvenience comes in. To do a multi-tape level-0 back-up, a person must attend to the outstanding tape requests and manually eject each tape when it fills up, and then insert the next tape in the series until the back-up is finally complete. This may mean someone has to periodically keep going to the computer room to check the console and see how far the back-up has gone, and this can become tiresome and inefficient.

Having a tape library (like an 8mm 7331-205 or 7331-305) where multiple tapes can be stacked ahead of time can prove very useful, but it still does not really help in the case of the **ontape** program, since the library does not know enough to eject the tape when the console message says to do so. A person must manually press the eject button and let the next tape load in.

The solution was to write a pair of cooperating scripts to perform the level-0 back-up unattended. The first script, infmx_lev0.sh, basically performs the **ontape** command specifying the level parameter desired. A **cron** entry can be made to **nohup** this script at the appropriate time/ day. As this script works, it logs its progress to an output log file. Then the partner script, infmx_lev0a.sh, watches the progress of the first job and takes the required action when necessary, ie ejecting each finished tape.

This technique can be utilized for other tasks as well, having one job ‘watch’ another and take action based on the progress of the first job.

Usually it is known how many tapes it takes to do the level-0, and the tapes can be labelled accordingly, eg 1 of 5, 2 of 5, etc.

A variable NUM_INFMX_TAPES is useful to check whether the expected number of tapes has been exceeded. Perhaps an increase in data may result in the full back-up requesting an additional tape, and using the variable will allow the script to know when to notify the system admin that the back-up is using more than the expected number of tapes, so that the labelling can be corrected and the new number of tapes can be staged for the next back-up. The variable should then be increased to reflect the new expected number of tapes required.

The script performs some initial checks for tape drive availability and ensures that the tape is not write-locked. It notifies the system admin if there are any problems, and then terminates. Once the **ontape** program is ready to be started, the first script submits the second watcher script to do what is necessary to get the back-up done in the least amount of time without the wasted time between tapes that often occurs while **ontape** waits for a person to load in the next tape.

The watcher script, infmx_lev0a.sh, will monitor the logfile of the first script, checking for when the ‘Please mount tape x’ message comes. What happens next is interesting. The **ontape** program apparently is written to check the tape label every few seconds to see whether or not the person has inserted the next tape in the sequence. While the current tape is being rewound and ejected, the **ontape** program will sometimes rapidly write out requests asking for the next tape to be inserted. I believe this is due to us ‘**nohup**ping’ the **ontape** progam in a script as opposed to running it at the system console. To minimize this behaviour, the **ontape** parent process is temporarily suspended. Once the rewind/eject has completed and the library has loaded the next sequential tape in the series, the parent process is awakened and then **ontape** proceeds.

Once a tape is inserted and **ontape** is writing to it, a default wait time is used so that the watcher waits a reasonable amount of time before scrutinizing the log for the expected mount request. This wait time can be specified as a parameter to the script, and defaults to 6,000 seconds (100 minutes). Once this time expires, the watcher script starts looking for the mount request once a minute, and every ten minutes it reports its progress with a timestamp message to its own log file so the system admin can see what it is doing.

The ejection of the tape can be tricky sometimes. The ejection request will often get an error about the tape drive being ‘temporarily unavailable’ or busy. To deal with this, the watcher script will perform the ejection routine as many as ten times, sleeping in between attempts, until it is successful. The timing has to be just right. It usually gets done in the first or second pass.

Finally, when the original script does complete the **ontape** back-up, it gets the PID of the watcher script and terminates it, since it is not required any longer. Then the completion status of the job is mailed to the system admin mail distribution list.

Sample output from the infmx_lev0.sh script is shown below:

```
*** Executing the /MBSYSMGR/mblogicals shell now...
Wed Apr 25 21:00:01 EDT 2001
infmx_drive is set to: rmt2
Using tapedrive: /dev/rmt2
***** infmx_lev0.sh procedure beginning at Wed Apr 25 21:00:01 EDT
2001....
Starting diagnostics
Testing rmt2
Testing scsi1
Testing sysplanar0
Ending diagnostics.
Submitting partner job /usr/mbprod/bin/infmx_lev0a.sh now...
    root 28566 43238   4 21:00:45      -  0:00 ksh /usr/mbprod/bin/
infmx_lev0a.sh
    root 43238 14724   2 21:00:01      -  0:00 ksh /usr/mbprod/bin/
infmx_lev0.sh
Wed Apr 25 21:00:45 EDT 2001

Please mount tape 1 on mb_rs1:/dev/rmt2 and press Return to continue ...
10 percent done.
20 percent done.
30 percent done.
40 percent done.
50 percent done.
```

```
Tape is full ...

Please label this tape as number 1 in the arc tape sequence.
This tape contains the following logical logs:

 15891

Please mount tape 2 on mb_rs1:/dev/rmt2 and press Return to continue ...
warning: previous tape is still mounted!
Please mount tape 2 on mb_rs1:/dev/rmt2 and press Return to continue ...
warning: previous tape is still mounted!
Please mount tape 2 on mb_rs1:/dev/rmt2 and press Return to continue ...
warning: previous tape is still mounted!
Please mount tape 2 on mb_rs1:/dev/rmt2 and press Return to continue ...
warning: previous tape is still mounted!
60 percent done.
70 percent done.
80 percent done.
90 percent done.
100 percent done.

Please label this tape as number 2 in the arc tape sequence.

Program over.
! Informix Level-0 Backup was successful !!
***** Backup completed at Wed Apr 25 23:39:43 EDT 2001....
***** Rewinding/Ejecting the tape....
***** Sending backup success mail at Wed Apr 25 23:40:02 EDT 2001....
*** Sendmail successful ***
Killing watcher process, PID is 28566...
***** infmx_lev0.sh procedure ending at Wed Apr 25 23:40:02 EDT 2001....
```

Sample output from the watcher job infmx_lev0a.sh script follows:

```
Wed Apr 25 21:00:45 EDT 2001
Wed Apr 25 21:00:55 EDT 2001: Sleeping initially for 6000 seconds...
(This is 100 minutes...)
Wed Apr 25 22:40:56 EDT 2001: Waiting for message to mount tape #2...
Wed Apr 25 22:40:56 EDT 2001: Now beginning to dynamically watch for
tape completion..
Wed Apr 25 22:40:56 EDT 2001: Found the message for tape no. 2..
Wed Apr 25 22:40:56 EDT 2001: Putting the ontape processes to sleep
while we change tapes...
Doing command: /usr/bin/tctl -f/dev/rmt2 rewoffl
Wed Apr 25 22:40:59 EDT 2001
Tape eject successful...
Wed Apr 25 22:41:18 EDT 2001
Waking the ontape processes back up again...
Wed Apr 25 22:44:21 EDT 2001: Sleeping initially for 6000 seconds...
(This is 100 minutes...)
```

The sample output shows a case where only two tapes were required to complete the back-up. This is due to using higher capacity 20/40GB tapes. When using 5GB or 7GB tapes, the same procedure could easily use five or six tapes to complete and that's when these scripts can really be useful to help the back-up to fully complete in the shortest possible time with no wasted time in between tape mounts.

INFMX_LEV0.SH

```
#!/bin/ksh
#
# infmx_levØ.sh
#
# Author: Michael Stanton
# stanton@mbusa.com
#
# Function: This script will perform a level-Ø full Informix
#           back-up using the Ontape utility. It works in conjunction
#           with a partner job (infmx_levØa.sh) in order to eject the
#           tapes as required for the multi-tape series.
#
# Modification History:
#======================
#  Date      Who        What
#  ----      ---        ----
#
#=================================================================
#
SEND_MAIL () {

    MAIL_FILE=/MAIL_API/api$$.bdy

    case $1 in
    "SUCCESS")
        echo "                                                    " >
$MAIL_FILE
        echo " ${place} Informix Level-Ø Backup was successful.. " >>
$MAIL_FILE
        echo "                                                    " >>
$MAIL_FILE
        echo "Message Source: $CURR_SHELL                         " >>
$MAIL_FILE
        MAIL_SUBJECT="${place} Informix Level-Ø backup successful !"
        ;;

    "EJECT")
        echo "                                                    " >
```

```
$MAIL_FILE
        echo " ${place} Informix Level-0 Backup was successful"    >>
$MAIL_FILE
        echo " but there was an error ejecting the tape !!!      " >>
$MAIL_FILE
        echo "                                                    " >>
$MAIL_FILE
        echo "Message Source: $CURR_SHELL                         " >>
$MAIL_FILE
        MAIL_SUBJECT="${place} Informix Level-0: Error ejecting tape!"
        ;;

    "ERROR")
        echo "                                                    " >
$MAIL_FILE
        echo " ! ! ! ! Error during Informix backup  ! ! ! !     " >>
$MAIL_FILE
        echo "                                                    " >>
$MAIL_FILE
        echo "Please investigate and correct ASAP .....          " >>
$MAIL_FILE
        echo "                                                    " >>
$MAIL_FILE
        echo "Message Source: $CURR_SHELL                         " >>
$MAIL_FILE
        MAIL_SUBJECT="${place} INFORMIX BACKUP ERROR - PROCESSING
TERMINATED"
        ;;

    "WRITE-LOCKED")
        echo "                                                    " >
$MAIL_FILE
        echo " ! ! The Lev0 tape is write-protected ! !          " >>
$MAIL_FILE
        echo "                                                    " >>
$MAIL_FILE
        echo "Please investigate and correct ASAP .....          " >>
$MAIL_FILE
        echo "                                                    " >>
$MAIL_FILE
        echo "Message Source: $CURR_SHELL                         " >>
$MAIL_FILE
        MAIL_SUBJECT="${place} Tape is write-locked- cannot do Lev0
backup!"
        ;;

    "NOTAPE")
        echo "                                                    " >
$MAIL_FILE
        echo " ! ! No tape in drive for the Informix backup  ! ! " >>
```

```
$MAIL_FILE
        echo "                                                    " >>
$MAIL_FILE
        echo "Please investigate and correct ASAP .....           " >>
$MAIL_FILE
        echo "                                                    " >>
$MAIL_FILE
        echo "Message Source: $CURR_SHELL                         " >>
$MAIL_FILE
        MAIL_SUBJECT="${place} No tape in drive to do Lev0 backup!"
        ;;

    *)
        echo "                                                    " >
$MAIL_FILE
        echo "Invalid parameter passed to the SEND_MAIL function." >>
$MAIL_FILE
        echo "                                                    " >>
$MAIL_FILE
        echo "Please investigate and correct...                   " >>
$MAIL_FILE
        echo "                                                    " >>
$MAIL_FILE
        echo "Message Source: $CURR_SHELL                         " >>
$MAIL_FILE
        MAIL_SUBJECT="$CURR_SHELL : INVALID PARAMETER - PROCESSING
TERMINATED"
        ;;
    esac

    mail -s "${MAIL_SUBJECT}"  ${TO_LIST}  <  ${MAIL_FILE}

    return
}
#
# **********************************************************************
# Start of main shell body
# **********************************************************************
#
    CURR_SHELL="$(basename $0)"
    backup_status="success"
    misc_options=""
    place=$(hostname)
    to_list="BACKUP_NOTIFY_DIS"
                                  # /etc/aliases mail distribution list

#...Since CRON'd jobs do not execute the variables file, call
mblogicals..
    echo "*** Executing the /MBSYSMGR/mblogicals shell now..."
    . /MBSYSMGR/mblogicals           # Assign any important variables
```

```
#...Use the tape drive defined in the variables file mblogicals above…
    echo "infmx_drive is set to: ${infmx_drive}"      #normally rmt1
    tapedrive="${infmx_drive}"
                            # This is the lower tape drive in the 7331

    print "Using tapedrive: /dev/${tapedrive}"
#
    echo "***** ${CURR_SHELL} procedure beginning at $(date)...."
#
#...Check if there is a tape inserted in the drive before we proceed...

#...Utilize Sysback's sbdevice command. If you do not have Sysback
#...installed,
#...then use the command tctl -f /dev/${tapedrive} rewind

    /usr/sbin/sbdevice -n /dev/${tapedrive}
                                      #see if tape is available or not
    if [ $? -ne 0 ]
    then
      SEND_MAIL NOTAPE
      exit 1
                  # terminate this job since we have no tape to use...
    fi

#...Check if the tape is writeable before we proceed...
#...Utilize Sysback's sbdevice command. If you do not have Sysback
#...installed,
#...then use the command tctl -f /dev/${tapedrive} write
    /usr/sbin/sbdevice -nw /dev/${tapedrive}
                                    #see if tape is write-locked or not
    if [ $? -ne 0 ]
    then
      SEND_MAIL WRITE-LOCKED
      exit 1
                   # terminate job since we cannot write to the tape...
    fi

#...Do a quick diag on the tape drive
    diag -cd ${tapedrive}

#...Spawn off a partner process whose purpose is to monitor this
#   job's logfile and take action when Ontape is requesting the next
#   tape. The partner job will issue the tctl eject command
#   and the 7331 library will then load the next tape in the series.

    echo "Submitting partner job /usr/mbprod/bin/infmx_lev0a.sh now..."
    nohup ksh /usr/mbprod/bin/infmx_lev0a.sh > /backupdir/
infmx_lev0a.log 2>&1 &
```

```
#...Show the running infmx_levØ* jobs..
    ps -ef | grep infmx_levØ | grep -v grep

    date

#...Do the Informix Level-Ø backup using the ontape utility
#...specifying the level-Ø parameter on the command line
#...This script will spend most of its time executing the next command…

    cat /dev/null | /usr/informix/bin/ontape -s -L Ø
                                                                  #start the backup

    if [ $? -ne Ø ]            # check the status after backup completes
    then
        echo "! ! ! !  Error during Informix Level-Ø backup at $(date)
! ! ! ! "
        backup_status="failure"
        SEND_MAIL ERROR
    else
        echo "! Informix Level-Ø Backup was successful !!"
        echo "***** Backup completed at $(date)...."

#...If backup was successful, rewind/eject the tape

      echo "***** Rewinding/Ejecting the tape...."
        /usr/bin/tctl -f/dev/${tapedrive} rewoffl            #auto eject
      if [ $? -ne Ø ]
      then
        SEND_MAIL EJECT                             #send mail and
           backup_status="failure"                  #try once more...
           /usr/bin/tctl -f/dev/${tapedrive}   rewoffl      #auto eject
      fi
    fi

#...Send success mail if backup worked.
    if [ "$backup_status" = "success" ]
    then
        echo "***** Sending backup success mail at $(date)...."
        SEND_MAIL SUCCESS
    fi

#...Now check and see if the watcher job, infmx_levØa.sh, is still
#      running looking for tape mount requests from this job. If we
#      find it (we should), then just kill that job since it no longer
#      serves any useful purpose now that we are done with the level-Ø
#      backup.

      WATCHER_PID=$(ps -ef |grep "infmx_levØa.sh" \
|grep -v grep|awk '{print $2}')
```

```
      if [[ ! -z ${WATCHER_PID} ]]
      then
            print "Killing watcher process, PID is ${WATCHER_PID}..."
            kill -KILL ${WATCHER_PID}
      fi

      ps -elf | grep ${WATCHER_PID} | grep -v grep
                                                   #should be gone now


    echo "***** ${CURR_SHELL} procedure ending at $(date)...."
#(eoj)
```

INFMX_LEV0A.SH

```
#!/bin/ksh
#
# infmx_levØa.sh
#
# Author: Michael Stanton
# stanton@mbusa.com
#
#
# Function: This is the partner job for infmx_levØ.sh. Its function
#           is to monitor the infmx_levØ logfile to determine when
#           Informix is requesting the next tape in the series. The
#           Level-Ø series currently takes six 8mm tapes to finish.
#           This job searches for the tape mount requests, and then
#           issues the /usr/bin/tctl command to rewind/eject the
#           tape. The 7331 tape library then places the tape back
#           into the proper slot, and gets the next tape in order.
#
# Note:   The infmx_levØ.sh script will be doing the final tape eject
#         of the series. This script will loop until
#         the lev-Ø procedure kills it when it no longer serves any
#         purpose.
#
# Modification History:
#=======================
#  Date      Who        What
#  ----      ---        ----
#=======================================================================
#
function SEND_MAIL {

    MAIL_FILE=/MAIL_API/api$$.bdy

    case $1 in
    "NOLOG")
```

```
        echo "                                                    " >
$MAIL_FILE
        echo " ${place} No level-Ø logfile to scan "                >>
$MAIL_FILE
        echo "                                                    " >>
$MAIL_FILE
        echo "Message Source: $CURR_SHELL                         " >>
$MAIL_FILE
        MAIL_SUBJECT="${place} Infmx Level-Ø partner job: no logfile!"
        ;;

    "NOPARENT")
        echo "                                                    " >
$MAIL_FILE
        echo " ${place} No PID found for ontape -s !"                >>
$MAIL_FILE
        echo "                                                    " >>
$MAIL_FILE
        echo "Message Source: $CURR_SHELL                         " >>
$MAIL_FILE
        MAIL_SUBJECT="${place} Infmx Level-Ø partner job: no ontape -s
process!"
        ;;

    "TOO_MANY_TAPES")
        echo "                                                    " >
$MAIL_FILE
        echo " ${place} The number of tapes has grown !           " >>
$MAIL_FILE
        echo "Expected no. of tapes is: ${NUM_INFMX_TAPES}.       " >>
$MAIL_FILE
        echo "Informix just took tape number: ${tapenum}.         " >>
$MAIL_FILE
        echo "                                                    " >>
$MAIL_FILE
        echo "Please see to it that the tapes are appropriately  " >>
$MAIL_FILE
        echo "labeled and correctly stacked to deal with this !  " >>
$MAIL_FILE
        echo "                                                    " >>
$MAIL_FILE
        echo "Message Source: $CURR_SHELL                         " >>
$MAIL_FILE
        MAIL_SUBJECT="${place} Lev-Ø backup is using more tapes than
expected!"
        ;;

    *)
        echo "                                                    " >
$MAIL_FILE
```

```
        echo "Invalid parameter passed to the SEND_MAIL function." >>
$MAIL_FILE
        echo "                                                    " >>
$MAIL_FILE
        echo "Please investigate and correct...                   " >>
$MAIL_FILE
        echo "                                                    " >>
$MAIL_FILE
        echo "Message Source: $CURR_SHELL                         " >>
$MAIL_FILE
        MAIL_SUBJECT="$CURR_SHELL : INVALID PARAMETER - PROCESSING
TERMINATED"
        ;;
    esac

    mail -s "${MAIL_SUBJECT}" ${to_list} < ${MAIL_FILE}

    return
}

function CHK_PARENT {

ONTAPE_PID=$(ps -ef |grep "ontape -s" |grep -v grep|awk '{print $2}')
SCRIPT_PID=$(ps -ef |grep "infmx_levØ.sh" |grep -v grep|awk '{print $2}'

      if [ -z ${ONTAPE_PID} ]
      then
            SEND_MAIL NOPARENT
            export PARENT_GONE="yes"

      fi

      return
}
#
# **********************************************************************
# Start of main shell body
# **********************************************************************
#
      CURR_SHELL="$(basename $Ø)"
      misc_options=""
      export PARENT_GONE="no"
      place=$(hostname)
      date

#...Call the variables definition script which includes the current
#...definition of the NUM_INFMX_TAPES variable (how many tapes we expect
#...the backup to use) as well as the infmx_drive variable for which
#...tape drive to use.
```

```
      . /MBSYSMGR/mblogicals

      logfile=/backupdir/infmx_lev0.log

#...Make sure there is a logfile there to scan...
      if [ ! -f ${logfile} ]
      then
            SEND_MAIL NOLOG
            exit 1
      fi
#
#
#...Set an integer variable for the tape number to watch for
#...If $1 is passed, we will take that to be the tape to look for
#...initially.
      param=$1
      typeset -i tapenum=${param:-2}
                                 #(we skip tape number 1 on purpose)

#...Set an integer variable for the initial time to wait after a tape
#...has been inserted. The default is 6000 seconds. If parameter $2 is
#...passed to this script, then it will use the value of $2 instead..

      timetowait=$2
      typeset -i initial_sleep=${timetowait:-6000}
      typeset -i calc_minutes=$((${initial_sleep}/60))

#...Set variables to control the eject of the tape looping code
      typeset -i num_eject_loops=10           #max tries
      typeset -i eject_loop_sleep=30     #allow 30 seconds
      typeset -i load_tape_sleep=180     #allow 3 minutes

#...Set a string variable to indicate how long initial sleep will be
      typeset -s first_time_thru="true"

#...Give the ontape -s process a boost in nice priority
      sleep 10
      CHK_PARENT                 #get the ontape -s PID first
      renice -n -20 -p ${ONTAPE_PID}

#...Loop looking for the infmx_lev0.sh's "please mount" messages...

      while true
      do

#...Call the CHK_PARENT routine to ensure that the ontape -s process
#...is still there...
            CHK_PARENT
            if [[ "${PARENT_GONE}" = "yes" ]]
            then
```

```
               print "$(date): Parent ontape process is gone,
exiting..."
               exit 1            #exit procedure if parent job goes away
            fi

            grep -q "Please mount tape ${tapenum}" ${logfile}
            if [ $? -eq Ø ]
                              #if we did find that message request...
            then

#...see if the number of tapes used has exceeded the expected number.
#...If so, send mail...
                  if [ ${tapenum} -gt ${NUM_INFMX_TAPES} ]
                  then
                        SEND_MAIL TOO_MANY_TAPES
                  fi

                  echo "$(date): Found the message for tape no.
${tapenum}.."

#...While we change tapes, we have to put the process to sleep. The
#...reason is that the ontape Informix utility continuously reads the
#...tape label to see if the tape that it's requesting has been
#...inserted. During the time the tape is being rewound it generates
#...error messages very rapidly since it
#...cannot read the tape label, so in order to keep it calm we 'stop'
#...the process temporarily during the switching of the tapes.

                  echo "$(date): Putting the ontape processes to sleep
while we change tapes..."
# (debug)            echo "PID of the ontape -s process is:
${ONTAPE_PID}"
# (debug)            echo "PID of its parent shell script
(infmx_levØ.sh) process is: ${SCRIPT_PID}"

# (debug)
             echo "Displaying the processes before suspending them..."
# (debug)            ps -elf |grep ${SCRIPT_PID} |grep -v grep

                  kill -STOP ${SCRIPT_PID}
                  sleep 3
                  kill -STOP ${ONTAPE_PID}

#                  echo "Displaying the suspended processes (note the T
in the 2nd column)..."
#                  ps -elf |grep ${SCRIPT_PID} |grep -v grep

#...Now we will eject the current tape from the drive...

                  echo "Doing command: /usr/bin/tctl -f/dev/
```

```
${infmx_drive} rewoffl"
                  eject_status="unknown"

#...track how many times we might fail attempting to eject the tape
#...since we can sometimes get 'device not currently available'.

                  typeset -i passes=0

                  until [ "${eject_status}" = "good" ]
                  do
                        date
                        /usr/bin/tctl -f/dev/${infmx_drive} rewoffl
                        if [ $? -eq 0 ]
                        then
                              eject_status="good"
                              echo "Tape eject successful..."
                        else
                          passes=$((passes + 1))
                          echo "$(date): Trying again to eject the tape"
                          echo "Sleeping ${eject_loop_sleep} seconds"
                          sleep ${eject_loop_sleep}
                          if [ ${passes} -gt ${num_eject_loops} ]
                          then
                              echo "***Can't get to the tape drive***"
                              echo "***Waking up the processes again***"
                     echo "Waking the ontape processes back up again..."
                                    kill -CONT ${ONTAPE_PID}
                                    sleep 3
                                    kill -CONT ${SCRIPT_PID}
                              fi
                        fi
                  done

                  date
                  sleep ${load_tape_sleep}
                             #give 7331 library time to load next tape

                  if [ ${passes} -le ${num_eject_loops} ]
                  then
                     echo "Waking the ontape processes back up again..."
                        kill -CONT ${ONTAPE_PID}
                        sleep 3
                        kill -CONT ${SCRIPT_PID}
#                        echo "Displaying the awoken ontape processes
(note A in the 2nd col)..."
#                        ps -elf |grep ${SCRIPT_PID} |grep -v grep
                  fi

                  tapenum=$((tapenum + 1))
                  first_time_thru="true"
            else
                  if [[ "$first_time_thru" = "true" ]]
                  then
                     echo "$(date): Sleeping initially for
${initial_sleep} seconds..."
                     echo "(This is ${calc_minutes} minutes...)"

                     sleep ${initial_sleep}
                     first_time_thru="false"
                     echo "$(date): Waiting for message to mount tape
#${tapenum}..."
                     echo "$(date): Now beginning to dynamically watch
for tape completion.."
                  else
                              #every 10 minutes, display a message..
                     case $(date +%M) in
                       "00"|"10"|"20"|"30"|"40"|"50")
                             echo "$(date): Waiting for msg to mount
tape #${tapenum}..."
                             sleep 60
                   #sleep another 1 minute. Report status each 10 mins.
                           ;;
                       *)
                           ;;
                     esac
                     sleep 5      #sleep 5 seconds before the next pass
                  fi
            fi
      done
#
#eoj
```

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

AIX news

Storix Software has announced Version 3.1 of Storix Backup Administrator for AIX. SBA provides everything from single directory to complete system back-ups (including multiple volume groups and raw logical volumes) and complete back-up management using a GUI. There is also a free SMIT-based back-up software option available for local system back-ups.

SBA provides centralized back-up administration for networked systems, stand-alone systems, and SP nodes.

For further information contact:
Storix Software, 3707 5th Avenue, Suite 125, San Diego, CA 92103, USA.
Tel: (619) 291 4500.
URL: http://www.storix.com.

* * *

Software AG has announced that its Tamino XML database product has been extended to include support for AIX and OS/390.

The product is designed to allow the use of XML to gain access to existing applications running on these platforms.

This support adds to existing Tamino support on Linux, Solaris, UnixWare, and Windows NT and 2000.

For further information contact:
Software AG, Uhlandstr 12, 64297 Darmstadt, Germany.
Tel: (6151) 92 0.
URL: http://www.softwareag.com/corporat/news/may2001/tamino_ibm_os390_aix.htm.

* * *

iPlanet E-Commerce Solutions has begun shipping Version 5.0 of its iPlanet Directory Server for secure user management services. The LDAP-based directory centralizes the process of account provisioning, management, and deletion across the enterprise and extranet, unifying identity management for all users.

Features include multi-master replication, which provides highly-available directory services. When a first server goes off-line, another is continuously available to store modified data. When the first server comes back on-line, it receives the modifications made while it was down.

Supported platforms include AIX, HP-UX, Solaris, and Windows NT and 2000.

For further information contact:
iPlanet, 901 San Antonio Rd, Palo Alto, CA 94303, USA.
Tel: (650) 960 1300.
URL: http://www.iplanet.com.

* * *

IBM has announced Version 3.6 of its WebSphere Application Server, Enterprise Edition, with a new release of Component Broker.

Also included is VisualAge component development, with tools to support application development on the AIX and NT platforms and targeting AIX, NT, Solaris, and OS/390.

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