71

AIX update

September 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

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.

Useful scripts for investigating space problems

Space problems are a common occurrence because applications are forever increasing in size. Often space problems can be caused by rogue processes or just careless users. Often their cause is obvious, but occasionally it can be a bit of a mystery.

After investigating numerous space problems over the years I have written the following scripts to ease the problem.

The first script is *dfs*. All this script does is sort your filesystems into order of percentage space used. This immediately shows you where the problem is.

Here is the *dfs* script:

```
#!/usr/bin/ksh
#
# Script: dfs
# Author: Roger Wickings
# Aim:    Show filesystems sorted by percentage full
#
awk="/usr/bin/awk"
basename="/usr/bin/basename"
df="/usr/bin/df"
egrep="/usr/bin/egrep"
grep="/usr/bin/grep"
sort="/usr/bin/sort"
tail="/usr/bin/tail"
uname="/usr/bin/uname"
#
# Functions
#
initialization()
{
  for parm in $PARMS
  do
    case "$parm" in
      -k)  DFOPT="-k"    ;;
      *)   :              ;;
    esac
  done
  return
}

showAIXspace()
{
  $df $DFOPT |
    $tail +2 |
    $grep -v ":" |
    $grep "^/dev" |
    $awk '{
           space = substr($4,1,length($4)-1)
           inode = substr($6,1,length($6)-1)
           if ( space > inode )
           {
             print space, $0
           }
           else
           {
             print inode, $0
           }
          }' |
    $sort -t" " +0 -1 -n |
    $awk -v bs=$DFOPT 'BEGIN {
           if ( bs == "" )
           {
             print "Filesystem          512-blocks      Free   %Used
Iused  %Iused Mounted on"
           }
           else
           {
             print "Filesystem         1024-blocks      Free   %Used
Iused  %Iused Mounted on"
           }
          }
          {
    printf "%-20s %9s %9s %7s %7s %7s %-s\n", $2, $3, $4, $5, $6, $7, $8
          }'
  return
}

showSpace()
{
  OS=`$uname `
  case "$OS" in
    AIX)   showAIXspace    ;;
    SunOS) showSUNspace    ;;
    *)     :               ;;
  esac
  return
}

showSUNspace()
{
  $df -k |
    $tail +2 |
    $grep -v ":" |
    $egrep -e "^/dev/|^swap" |
    $awk '{
           space = substr($5,1,length($5)-1)
           print space, $0
          }' |
    $sort -t" " +0 -1 -n |
    $awk 'BEGIN {
             print "Filesystem              1024-blocks      Used
Avail   %Used  Mounted on"
          }
          {
           printf "%-25s %9s %9s %9s %7s %-s\n", $2, $3, $4, $5, $6, $7
          }'
  return
}
#
# Start of main processing
#
SCRIPT=`$basename $0 `
PARMS="$*"

initialization
showSpace

exit 0
```

The output from this script is shown below:

```
Filesystem          512-blocks      Free   %Used   Iused  %Iused Mounted on
/dev/hd1                 32768     31528      4%      39      1% /home
/dev/homelv              16384     15792      4%      16      1% /home/
spnode09
/dev/proddatalv          16384     15784      4%      17      1% /prod_data
/dev/hd3                 81920     72968     11%     455      5% /tmp
/dev/optwgetlv          114688    101904     12%     177      2% /opt/wget
/dev/sudolv              81920     71432     13%     161      2% /opt/sudo
/dev/optlv               16384     11840     28%      48      3% /opt
/dev/ftipropdatalv       16384     10504     36%     330     17% /fti_refdata
/dev/hd9var             196608    108176     45%    1486      7% /var
/dev/tivoliv3.1lv      1851392   1021392     45%    6883   3% /opt/Tivoli_v3.1
/dev/ftilv              131072     55064     58%    3144     20% /fti
/dev/hd4                 65536     26760     60%    2110     13% /
/dev/hd2               1523712    425736     73%   24595     13% /usr
/dev/pgplv               65536     17624     74%      41      1% /opt/pgp
/dev/ftihomelv           49152      7816     85%    2872     47% /fti_home
/dev/messagelv         1015808       400    100% 748 1% /prod_data/message_log
```

Once I have identified the filesystem with the problem, my first method of attack is to determine the largest files within that filesystem. For this I use script *finds*. This script lists all the files within the filesystem in size order. I usually pipe the output through *tail* just to list the largest files, eg:

```
     finds /prod_data/message_log | tail -10
```

or:

```
     cd /prod_data/message_log
     finds . | tail -10
```

Here is the *finds* script:

```
#!/usr/bin/ksh
#
# Script: finds
# Author: Roger Wickings
# Aim:    List files in a filesystem by size
#
awk="/usr/bin/awk"
basename="/usr/bin/basename"
cut="/usr/bin/cut"
find="/usr/bin/find"
sort="/usr/bin/sort"
#
# Functions
#
checkParms()
{
  FILELIST=""
  for parm in $PARMS
  do
    if test "$parm" = "-r"
    then
      REVERSE="r"
    else
      check=`echo "$parm" | $cut -c1 `
      if test "$check" != "-"
      then
        FILELIST="$FILELIST $parm"
      fi
    fi
  done
  return
}
listFiles()
{
  $find $FILELIST -xdev -type f -ls |
    $awk '{ print $7 ": " $0 }' |
    $sort -t: +0 -1 -n$REVERSE |
    $awk '{ printf "%7s %4s %10s %2s %-8s %-8s %10s %3s %2s %5s %s\n",
$2, $3, $4, $5, $6, $7, $8, $9, $10, $11, $12 }'
  return
}
#
# Start of main processing
#
SCRIPT=`$basename $0 `
PARMS="$*"

checkParms
listFiles

exit 0
```

The output of script *finds* is shown below:

```
 821  476 -r--r--r--  1 root     sys          485645 Jun 23 23:55 ./
old_pid_log/tivoliLogAppendDaemon
 270  672 -rw-rw-rw-  1 root     sys          684393 Jun 25 15:32 ./
pid_log/ftpcopy.20010625
 834 1012 -r--r--r--  1 root     sys         1034940 Jun 24 23:59 ./
old_pid_log/ftpcopy.20010624
 855 1036 -r--r--r--  1 root     sys         8060455 Jun 23 23:59 ./
old_pid_log/ftpcopy.20010623
 820 1064 -r--r--r--  1 root     sys        10089188 Jun 22 23:59 ./
old_pid_log/ftpcopy.20010622
 801 1100 -r--r--r--  1 root     sys        12126066 Jun 21 23:59 ./
old_pid_log/ftpcopy.20010621
 669 1120 -r--r--r--  1 root     sys        15143563 Jun 20 23:59 ./
old_pid_log/ftpcopy.20010620
 712 1120 -r--r--r--  1 root     sys        21144474 Jun 18 23:59 ./
old_pid_log/ftpcopy.20010618
 746 1120 -r--r--r--  1 root     sys        21146134 Jun 19 23:59 ./
old_pid_log/ftpcopy.20010619
 693 3880 -r--r--r--  1 root     sys       343969323 Jun 23 23:55 ./
old_pid_log/ADSM_full_backup.20010623
```

If this does not immediately highlight the cause of the problem, my next line of attack is to list all the files in the filesystem sorted by modification time, in the hope that the files most recently changed are the cause of the problem. For this I use script *findt* listed below:

```
#!/usr/bin/ksh
#
# Script: findt
# Author: Roger Wickings
# Aim:    Display files in a filesystem sorted by modification time
#
```

```
awk="/usr/bin/awk"
basename="/usr/bin/basename"
cut="/usr/bin/cut"
date="/usr/bin/date"
find="/usr/bin/find"
sed="/usr/bin/sed"
sort="/usr/bin/sort"
#
# Functions
#
checkParms()
{
  FILELIST=""
  for parm in $PARMS
  do
    if test "$parm" = "-r"
    then
      REVERSE="-r"
    else
      check=`echo "$parm" | $cut -c1 `
      if test "$check" != "-"
      then
        FILELIST="$FILELIST $parm"
      fi
    fi
  done
  return
}

showFiles()
{
  checkParms

  (
    $date +"%Y %m %d"
    $find $FILELIST -xdev -type f -ls
  ) |
    $awk '
          NR == 1 {
                    thisyear  = $1
                    lastyear  = $1 - 1
                    thismonth = substr($2+100,2,2)
                    thisday   = substr($3+100,2,2)
                  }
          NR != 1 {
                    day = substr($9+100,2,2)

                    if ( $8 == "Jan" ) { month = "01" }
                    if ( $8 == "Feb" ) { month = "02" }
                    if ( $8 == "Mar" ) { month = "03" }
                    if ( $8 == "Apr" ) { month = "04" }
                    if ( $8 == "May" ) { month = "05" }
                    if ( $8 == "Jun" ) { month = "06" }
                    if ( $8 == "Jul" ) { month = "07" }
                    if ( $8 == "Aug" ) { month = "08" }
                    if ( $8 == "Sep" ) { month = "09" }
                    if ( $8 == "Oct" ) { month = "10" }
                    if ( $8 == "Nov" ) { month = "11" }
                    if ( $8 == "Dec" ) { month = "12" }

                    if ( length($10) == 4 )
                    {
                      year = $10
                      hour = "00:00"
                    }
                    else
                    {
                      if ( month <= thismonth ) { year = thisyear }
                      if ( month >  thismonth ) { year = lastyear }
                      hour = $10
                    }
                    print year "/" month "/" day, hour, $0
                  }
         ' |
    $sort $REVERSE |
    $cut -c17-

  return
}
#
# Start of main processsing
#
SCRIPT=`$basename $0 `
PARMS="$*"

showFiles

exit 0
```

The output from this script is identical in format to that of script *finds*.

Once you have discovered the cause of your space problem, it is best to copy the offending files elsewhere and then to clear them down, ie:

```
cat /dev/null > /filename
```

This has the advantage of freeing the disk space immediately, which removing a file will not do if the file is open to a running process.

Roger Wickings
Systems Programmer
FT Interactive Data (UK)

Check user security in AIX

The company I work with has some 15 AIX servers. On these servers reside lots of users. A userid is created whenever he or she does some work on a server. Sometimes people leave the company, which can cause problems with security.

The AIX administrator knows when somebody needs to have a userid and will create it. However, there are a lot of things the administrator does not know, for example:

- People who have changed roles or jobs.
- People who have left the company.
- Requests to login on certain servers which are more numerous than expected for the job.

All of these make the job of the administrator harder – he needs to know what all the people are doing at all sites in the company. This is not what the administrator's job was intended to be.

I wrote some scripts to make my job easier. When the scripts have been run I get lots of information about all the users. One of the scripts, *users_to_remove,* extracts data that gives the information about whether to drop or lock users. Another script, *users_alert,* shows me any gaps in security – no history of a password, a password of one character, etc.

AIX_USER

The script aix_user will gather the files we need to extract information from. These files are:

- /etc/group

  Here we can find the group/user combinations and group ID numbers.

- /etc/passwd

This contains users and the values userid, primary group ID, user information, and home directory.

- /etc/security/lastlog

  In this file we can get the time in seconds when a user has logged in.

- /etc/security/user

  This contains all the other variables that we need.

The next step is to extract all the data and put it in a file called *users.aix*.

At the end it will execute the scripts *users_to_remove* and *users_alert*.

You can gather all the information about all the servers if no parameters are specified, or gather all the information about just one server, by giving the servername as a parameter, eg:

```
aix_users server1.dom3.company.
```

The script can be started in the foreground/background or by using crontab.

```
#!/bin/ksh
# Name              : aix_user
# Last change       : 11-06-2001 T.W. Post - creation script
# Description       : gather all aix users and put them in a database
# Parameter         : optional : hostname
#                    if it exists: only that hostname will be processed and
#                     the users will be added to the outputfile users.aix
#                     You can use it if a site did not respond and
#                     when the server is started you can add the users
#                     without reloading all other servers.
#                     example : aix_user  - reload the entire database
#                               aix_user server1.dom1.company - add users
#                                            on this server to the database
#------------------------------------
echo "`date` : begin script"
#
# company_name used in the host parameter e.g.
#              HOST=aixserver.domain.company
# This variable contains only the last part
company_name=company

# ftp_user: user used for ftp and starting this script
ftp_user=ftp_user_id
```

```
# work_dir : Directory where we can find the files we need and
#            we can put the result
#      optional: also_valid.users
#      result  : users.aix
work_dir="/home/work_dir"

# password_dir      : directory used for password files
# password_filename : file name containing hosts + their
#                     corresponding passwords of hosts that do
#                     not have the general password of the ftp_user
password_dir="/home/password"
password_filename="ftp_passwords"

# general_dir : directory used for general file :
#               all_aix_servers  - containing all aix servers
general_dir="/home/general"

# End of company / site dependent variables
#
# ===================================================================

default="default"
cd $work_dir

# Save the old file
cp users.aix users.aix.old 2>/dev/null

only_one="all"

if [ $# -eq 1 ]
then echo
"*****************************************************************"
     echo "*"
     echo "* Only data of server $1 will be processed and "
     echo "* added to the file users.aix "
     echo "*"
     echo
"*****************************************************************"
     server=$1
     only_one="one"
     general_dir="/tmp"
     echo "$server" >/tmp/all_aix_servers
else >users.aix
     fi

# Get the ftp password out of a passwordfile
ftp_password='cat $password_dir/$password_filename 2>/dev/null'
if [ "$ftp_password" ]
then echo "Password of ftp user was found, continuing ...\n"
     fi
     # removing old entries just to be sure there is no old stuff
     rm user.*.$company_name*     2>/dev/null
     rm passwd.*.$company_name*   2>/dev/null
     rm lastlog.*.$company_name*  2>/dev/null
     rm group.*.$company_name*    2>/dev/null
     rm ping_*.*.$company_name*   2>/dev/null
     rm ftp_*.*.$company_name*    2>/dev/null
     >not_responding_sites

     # Get all the files we need
     for site in 'cat $general_dir/all_aix_servers'
        do
        # if a site doesn't have the standard password, use the password
        # mentioned in this file
        if [ -s $password_dir/$password_filename.$site ]
        then password='cat $password_dir/$password_filename.$site'
        else password=$ftp_password
             fi

        # build the .netrc file to get tnsnames.ora and the oratab
        echo "machine $site login $ftp_user password $password" \
              >/tmp/all_aix.netrc
     echo "macdef init"                             >>/tmp/all_aix.netrc
     echo "get /etc/group       group.$site"        >>/tmp/all_aix.netrc
     echo "get /etc/passwd passwd.$site"            >>/tmp/all_aix.netrc
     echo "get /etc/security/user user.$site"       >>/tmp/all_aix.netrc
     echo "get /etc/security/lastlog lastlog.$site" >>/tmp/all_aix.netrc
        echo "quit"                                 >>/tmp/all_aix.netrc
        echo ""                                     >>/tmp/all_aix.netrc
        cp /tmp/all_aix.netrc    $HOME/.netrc
        chmod 600                $HOME/.netrc
        # .netrc file is now ok, we are ready to ftp

        # check if host is online otherwise the FTP is not responding
        ping -c3 $site 1>ping_$site.result 2>&1
        if [ $? -eq 0 ]
        then # site has been reached : ftp can be started
             ftp $site 1>/dev/null 2>&1
             rm ping_$site.result
        else echo $site >>not_responding_sites
           echo "$site did not respond, script continuing with warnings"
             fi
        done

# remove the netrc file + the help file
rm $HOME/.netrc              2>/dev/null
rm /tmp/all_aix.netrc

#
```

```
# Set date to check in amount_at_least in seconds one year ago
#          eg 1 jun 2001 -> 1 jun 2000
# amount of 1 jan 2000 00:00:00  : 926681200
subtract_at_least=946681200
year_2000=946681200

# Get current year
Year_at_least='date +%Y'

# Get amount of seconds in this year
julian_day='date +%j'
let julian_sec=$julian_day*86400

# Make check year last year
let Year_at_least=$Year_at_least-1

# Add amount of seconds of total amount of seconds to skip (
subtract_at_least )
year_from=2000
while [ $year_from -lt $Year_at_least ]
     do
     # Check for leap year
     let year_help=year_from/4
     let year_help=year_help*4
     if [ $year_help -eq year_from ]
     then # Leap year
          let subtract_at_least=$subtract_at_least+31622400
     else # Normal year
          let subtract_at_least=$subtract_at_least+31536000
          fi
     let year_from=$year_from+1
     done

# Add current seconds to previous year
let subtract_at_least=$subtract_at_least+$julian_sec

# Now we have the amount in seconds 1 year ago in subtract_at_least
# we can use it in the next loop

# Process all users / server
for file in 'ls passwd*'
     do
     server='echo $file | cut -f2- -d"."'
     echo "`date` : Processing server $server"
     cat $file |while read line
          do
          user='echo $line | cut -f1 -d":"'
          user_id='echo $line | cut -f3 -d":"'
          prim_group_id='echo $line | cut -f4 -d":"'
          user_info='echo $line | cut -f5 -d":"'
          home='echo $line | cut -f6 -d":"'
     prim_group='fgrep ":$prim_group_id:" group.$server | cut -f1 -d":"'

          # get all groups of user out of group.server
          cat group.$server | grep -E ",$user|:$user" >temp.groups
          group=""
          cat temp.groups | while read line2
               do
               group="$group 'echo $line2 | cut -f1 -d":"'"
               done

          break="off"
          # process user file
          cat user.$server | grep -v "^\*" |while read parm is value
              do
              # Use default user to grab all defaults
              # Override them with user values
              if [ "$parm" = "$default:" -o "$parm" = "$user:" ]
              then check_case="1"
                   if [ "$parm" = "$user:" ]
                   then break="on"
                        # Break if user is processed
                        fi
              else check='echo $parm | grep ":"'
                   # Check if we have a new user section
                   if [ -n "$check" ]
                   then check_case="Ø"
                        if [ "$break" = "on" ]
                      then # reset break for the next time and break out
                             break="off"
                             break
                             fi
                        fi
                    fi

               # We have a parameter, check if it is the user we need
               # and set the value
               if  [ "$check_case" = "1" ]
               then case $parm in
                         "account_locked" ) account_locked=$value ;;
                         "login" )      login=$value ;;
                         "rlogin" )     rlogin=$value ;;
                         "su" )         su=$value ;;
                         "sugroups" )   sugroups=$value ;;
                         "expires" )    expires=$value ;;
                         "histexpire" ) histexpire=$value ;;
                         "histsize" )   histsize=$value ;;
                         "minage" )     minage=$value ;;
                         "maxage" )     maxage=$value ;;
                         "maxexpired" ) maxexpired=$value ;;
                         "minalpha" )   minalpha=$value ;;
                         "minother" )   minother=$value ;;
                         "minlen" )     minlen=$value ;;
                         "mindiff" )    mindiff=$value ;;
                         "maxrepeats" ) maxrepeats=$value ;;
                          *) ;;
                         esac
                    fi
               done

           # Initialize login date
           login_date="Not set"

           # get user data
          cat lastlog.$server | grep -v "^*"  | while read parm is value
               do
               if [ "$parm" = "$user:" ]
               then check_case="1"
                    if [ "$parm" = "$user:" ]
                    then break="on"
                         fi
               else check='echo $parm | grep ":"'
                    if [ -n "$check" ]
                    then check_case="Ø"
                         if [ "$break" = "on" ]
                         then break="off"
                              break
                              fi
                         fi
                    fi

              if  [ "$check_case" = "1" -a "$parm" = "time_last_login" ]
               then login_date=$value
                    # We have found the value!
                    break
                    fi
               done

          # Check if the user has logged in sometimes
          if [ "$login_date" != "Not set" ]
          then # calculate day of last login : the value is in seconds
               typeset -i year_help
               typeset -i month
               # Check one year ago
               let time_left=$login_date-$subtract_at_least
               if [ $time_left -lt Ø ]
               then # Display not a value more than 1 full year ago
                    login_date="more than 1 year ago"
               else # Get date
                    # Subtract years
```

```
year=2000
let time_left=$login_date-$year_2000
while [ $time_left -gt 0 ]
     do
     # Check for leap year
     let year_help=year/4
     let year_help=year_help*4
     if [ $year_help -eq year ]
     then subtract=31622400
          leap_year=Y
     else subtract=31536000
          leap_year=N
          fi
     let time_left=$time_left-$subtract
     let year=$year+1
     done

# One year to much subtracted
let time_left=$time_left+$subtract

# Subtract months, begin at januari
let month=1
while [ $time_left -gt 0 ]
     do
     case $month in
          1 ) seconds=2592000
              month_="jan"
              let year=$year-1
              prev_day="31 dec $year" ;;
          2 ) if [ "$leap_year" = "Y" ]
              then seconds=2505600
              else seconds=2419200
                   fi
              month_="feb"
              prev_day="31 jan $year" ;;
          3 ) seconds=2678400
              month_="mar"
              if [ "$leap_year" = "Y" ]
              then prev_day="29 feb $year"
              else prev_day="28 feb $year"
                   fi ;;
          4 ) seconds=2592000
              month_="apr"
              prev_day="31 mar $year" ;;
          5 ) seconds=2678400
              month_="may"
              prev_day="30 apr $year" ;;
          6 ) seconds=2592000
              month_="jun"
              prev_day="31 may $year" ;;
          7 ) seconds=2678400
              month_="jul"
              prev_day="30 jun $year" ;;
          8 ) seconds=2678400
              month_="aug"
              prev_day="31 jul $year" ;;
          9 ) seconds=2592000
              month_="sep"
              prev_day="31 aug $year" ;;
         10 ) seconds=2678400
              month_="oct"
              prev_day="30 sep $year" ;;
         11 ) seconds=2592000
              month_="nov"
              prev_day="31 oct $year" ;;
         12 ) seconds=2678400
              month_="dec"
              prev_day="30 nov $year" ;;
         13 ) seconds=2678400
              month_="dec"
              prev_day="31 dec $year" ;;
        * ) echo "ERROR: the end of time is reached"
         esac
     let time_left=$time_left-$seconds
     let month=$month+1
     done

# Correct month and time_left
let month=$month-1
if [ $month -eq 0 ]
then month=12
     let year=$year-1
     fi
let time_left=$time_left+$seconds

# Subtract days
day=0
while [ $time_left -gt 0 ]
     do
     let day=$day+1
     let time_left=$time_left-86400
     done

# Correct day
let day=$day-1
if [ $day -gt 0 ]
then login_date="$day $month_ $year"
else login_date=$prev_day
     fi
fi
```

```
               fi
          # Now we have all the values, save them all in a file
          echo "Processing user $user on server $server"
          echo "$user $server $user_id $prim_group $home $login $rlogin
$account_locked $su $histexpire $histsize $minage $maxage $maxexpired
$minalpha $minother $minlen $mindiff $maxrepeats $expires $group
|$user_info |$sugroups |$login_date" >>users.aix
          done
     done

# Remove stuff we needed before
rm group.*.$company_name*
rm lastlog.*.$company_name*
rm passwd.*.$company_name*
rm temp.groups               2>/dev/null
rm user.*.$company_name*

# Do not forget the temporary file
if [ "$only_one" = "one" ]
then rm /tmp/all_aix_servers
     fi

echo "\n\n`date` : Executing users_to_remove "
$work_dir/users_to_remove

echo "Output of users_to_remove can be found in $work_dir/remove.users"

echo "\n\n`date` : Executing users_alert"
$work_dir/users_alert

echo "Output of users_alert can be found in $work_dir/alerts.users"

if [  -s not_responding_sites ]
then echo "The next site(s) did not respond: "
     cat not_responding_sites
     echo "\nPlease check this sites, and restart aix_user \c"
     echo "with the site parameter for each site. "
     echo "example : aix_user site1.dom3.company"
else rm not_responding_sites
     fi

echo "`date` : end script"
# End script aix_user
```

USERS_TO_REMOVE

The script users_to_remove contains three parts:

- Users who can be removed at all sites.

- Users who can be removed at some sites.
- Users who are locked.

There are three possible reasons to get into this file: you never log in, you have not logged in in the past year, or you are locked.

If a user did not log in for a year I may assume he or she is doing something different from a year ago, but the initial password was set. For security reasons (if they do not log in, anyone else can enter the standard ‘easy’ password to log in) these users have to be locked at once. The next step is to inform Personnel to ask what to do with these users. If the user has changed his or her job or has left the company, I remove the user and their home directory. If the user is still working in the company I call her/him and ask if their userid can be dropped – mostly it can. If it is not possible to drop it, I lock their userid.

```
# Name           : users_to_remove
# Last change    : 11-06-2001 T.W. Post - creation script
# Description    : display users who did not login
#—————-————————————————————
# Set length of display variables
typeset -L12 disp_user
typeset -L25 disp_server
typeset -L21 disp_expired

Headline="Userid       Server                   Login date
login  rlogin"
# Get string of users you do not want to display
valid_users=""
cat also_valid.users 2>/dev/null | while read user
    do
    if [ "$valid_users" ]
    then valid_users="^$user |$valid_users"
    else valid_users="^$user"
         fi
    done

if [ ! "$valid_users" ]
then valid_users="non existing user"
     fi

# get user who has not logged on since the previous year with their
# corresponding servers
cat users.aix | grep -E "Not set|more than 1 year ago" | grep -v -E
"$valid_users" | sort | cut -f1,2 -d" " >users_to_revoke.tmp
```

```
# get only username of user who has not logged on
# since the previous year
string=""
cat users.aix | grep -E "Not set|more than 1 year ago" | grep -v -E
"$valid_users" | cut -f1 -d" " | sort -u | while read user
          do
          if [ "$string" ]
          then string="$user |$string"
          else string="$user "
               fi
          done

# get users which did logon on other servers
cat users.aix | grep -E "$string" | cut -f1,2 -d" " | sort -o
users_to_revoke.tmp2

# get users who did not logon on any server
diff users_to_revoke.tmp users_to_revoke.tmp2 | grep ">" | cut -f2 -d" "
| sort -u -o users_to_revoke.tmp3

string=""
# create a sting of them
cat users_to_revoke.tmp3 | while read user
    do
    if [ "$string" ]
    then string="^$user |$string"
    else string="^$user "
         fi
    done

echo "Users who never logged in on all servers: " >remove.users
echo "$Headline " >>remove.users
# display the users and their servers which can everwhere be revoked
cat users_to_revoke.tmp | grep -v -E "$string" >users_to_revoke.tmp4
cat users_to_revoke.tmp4 | while read user server
do   line='egrep "^$user $server" users.aix'
     expired='echo $line | cut -f4 -d"|"'
     login='echo $line | cut -f6 -d" "'
     rlogin='echo $line | cut -f7 -d" "'
     locked='echo $line | cut -f8 -d" "'
     expires='echo $line | cut -f20 -d" "'
     disp_user=$user
     disp_server=$server
     if [ "$expired" = "Not set" ]
     then expired="Never logged on"
          fi
     disp_expired=$expired
     # display only users who are able to log in
     if [ "$locked" = "false" -a \( "$login" = "true" -o "$rlogin" =
"true" \) ]
```

```
     then if [ "$expires" != "0101000070" ]
          then echo "$disp_user $disp_server $disp_expired $login
$rlogin" >>remove.users
               fi
          fi
     done

echo "\n\nUsers who can be partly revoked ( not on all servers) : "
>>remove.users
echo "$Headline " >>remove.users
diff users_to_revoke.tmp users_to_revoke.tmp4 | grep "^<" | cut -c2-
>users_to_revoke.tmp5
cat users_to_revoke.tmp5 | while read user server
do   line='egrep  "^$user $server" users.aix'
     expired='echo $line | cut -f4 -d"|"'
     login='echo $line | cut -f6 -d" "'
     rlogin='echo $line | cut -f7 -d" "'
     locked='echo $line | cut -f8 -d" "'
     expires='echo $line | cut -f20 -d" "'
     disp_user=$user
     disp_server=$server
     if [ "$expired" = "Not set" ]
     then expired="Never logged on"
          fi
     disp_expired=$expired

     # display only users who are able to log in
     if [ "$locked" = "false" -a \( "$login" = "true" -o "$rlogin" =
"true" \) ]
     then if [ "$expires" != "0101000070" ]
          then echo "$disp_user $disp_server $disp_expired $login
$rlogin" >>remove.users
               fi
          fi
     done

# remove work files
rm users_to_revoke.tmp
rm users_to_revoke.tmp2
rm users_to_revoke.tmp3
rm users_to_revoke.tmp4
rm users_to_revoke.tmp5

echo "\n\nUsers whose accounts are locked : " >>remove.users
echo "$Headline" >>remove.users
cat users.aix | while read user server a b c login rlogin locked rest
     do
     if [ "$locked" = "true" ]
     then disp_user=$user
          disp_server=$server
          expired='echo $rest | cut -f4 -d"|"'
          if [ "$expired" = "Not set" ]
          then expired="Never logged on"
               fi
          disp_expired=$expired
          echo "$disp_user $disp_server $disp_expired $login   $rlogin"
>>remove.users
          fi
     done

# end script users_to_remove
```

USER_ALERT

The users_alert script has just one part – to check whether all password variables are set. The following user variables will be checked: histexpire, histsize, minage, maxage, maxexpired, minalpha, minother, minlen, mindiff, and maxrepeats.

The defaults for these variables can be found in */etc/security/user* following the paragraph 'default:'. All users who have another value will be named after the default paragraph.

These entries are required to perform some security on an AIX server.

The variables used and when to use them is described below:

- histexpire – defines the period of time in weeks that a user will not be able to reuse a password. If not filled in, the user can use the same password again and again until the time ends.
- histsize – defines the number of previous passwords which cannot be reused. Use it because the user will define, for example, 10 passwords but will still use the same password he or she already had.
- minage – defines the minimum number of weeks between password changes. It prevents the user from changing his or her password several times within a minute.
- maxage – defines the maximum number of weeks a password is valid. It will make sure that a user does change the password.
- maxexpired – defines the maximum number of weeks after

maxage that an expired pasword can be changed by a user. It gives the user some time to change their password, after which the administrator has to do the job.

- minalpha – defines the minimum number of alphabetic characters in a password. A password of ‘a’ is easy to remember.
- minother – defines the minimum number of non-alphabetic characters in a password. A password containing a word is easy to decrypt.
- minlen – defines the minimum length of a password. A password of one character can be typed in a second – by anyone!
- mindiff – defines the minimum number of characters in the new password that did not occur in the old password. If you had a password of John1 in January you might change it to John2 in February. This is easy to crack.
- maxrepeats – defines the maximum number of times a given character can appear in a password. The password aaaaa1 is not a ‘real’ password, so forbid it.

```
# Name            : user_alert
# Last change     : 11-06-2001 T.W. Post - creation script
# Description     : display various default variables which had to be set
#————————————————————————————————————
>alerts.users
cat users.aix | while read user server a b c login rlogin locked g
histexpire histsize minage maxage maxexpired minalpha minother minlen
mindiff maxrepeats expires rest
     do
     # display only users who are able to log in
     if [ "$locked" = "false" -a \( "$login" = "true" -o "$rlogin" =
"true" \) ]
     then if [ "$expires" != "0101000070" ]
          then if [ $histexpire -eq 0 -o $histsize -eq 0 -o $minage -eq
0 -o $maxage -eq 0 -o $maxexpired -lt 1 -o $minalpha   -eq 0 -o
$minother   -eq 0 -o $minlen     -eq 0 -o $mindiff    -eq 0 -o
$maxrepeats -eq 8 ]
              then echo "$user $server alert : "         >>alerts.users
                   if [ $histexpire -eq 0 ]
                   then echo "          histexpire = 0 " >>alerts.users
                        fi
                   if [ $histsize -eq 0 ]
                   then echo "          histsize = 0 "   >>alerts.users
```

```
                    fi
               if [ $minage -eq 0 ]
               then echo "          minage = 0 "     >>alerts.users
                    fi
               if [ $maxage -eq 0 ]
               then echo "          maxage = 0 "     >>alerts.users
                    fi
               if [ $maxexpired -lt 1 ]
       then echo "          maxexpired = $maxexpired ">>alerts.users
                    fi
               if [ $minalpha -eq 0 ]
               then echo "          minalpha = 0 "   >>alerts.users
                    fi
               if [ $minother -eq 0 ]
               then echo "          minother = 0 "   >>alerts.users
                    fi
               if [ $minlen -eq 0 ]
               then echo "          minlen = 0 "     >>alerts.users
                    fi
               if [ $mindiff -eq 0 ]
               then echo "          mindiff = 0 "    >>alerts.users
                    fi
               if [ $maxrepeats -eq 8 ]
               then echo "          maxrepeats = 8 " >>alerts.users
                    fi
               fi
          fi
      fi
 done

# end script user_alert
```

GET_USER

You can use the script get_user to get all the information about one user. It will produce one page per AIX machine. You can start it with or without the parameter ': userid'. If you start without it, the script will ask you for a userid.

```
# Name              : get_user
# Last change       : 11-06-2001 T.W. Post - creation script
# Description       : Display an AIX user on all servers he or she resides
# Parameter         : optional : username
#                     if it does not exist : the script asks for the user
#                     example get_user user123
#————————————————————————————————————-
# Set type of numeric values, it is better to look at
typeset -R3   histexpire
typeset -R3   histsize
typeset -R3   minage
typeset -R3   maxage
typeset -R3   maxexpired
typeset -R3   minalpha
typeset -R3   minother
typeset -R3   minlen
typeset -R3   mindiff
typeset -R3   maxrepeats

# Get the user if it is not given
if [ $# -eq 0 ]
then echo "Userid to display : \c"
     read user
else user=$1
     fi

if [ ! "$user" ]
then echo "We need an userid, script exiting ... "
     exit 1
     fi

# We need to save the user, because the read in the while destroys
# this value
save_user=$user

cat users.aix | grep "^$user "  | while read user server user_id
prim_group home login rlogin account_locked su histexpire histsize
minage maxage maxexpired minalpha minother minlen mindiff maxrepeats
rest
     do
     clear
     group_set='echo $rest | cut -f1 -d"|"'
     user_info='echo $rest | cut -f2 -d"|"'
     sugroups='echo $rest | cut -f3 -d"|"'
     lastlogin='echo $rest | cut -f4 -d"|"'

     if [ "$lastlogin" = "Not set" ]
     then lastlogin="Never logged in"
          fi

     # Display the user
     echo "Info of server   : $server "
     echo "user             : $user "
     echo "user id          : $user_id "
     echo "primary group    : $prim_group "
     echo "groups           : $group_set "
     echo "home directory   : $home "
     echo "user information : $user_info "
     echo "login            : $login "
     echo "rlogin           : $rlogin "
     echo "locked account   : $account_locked "
     echo "last login       : $lastlogin "
     echo "su               : $su "
     echo "sugroups         : $sugroups "
     echo "histexpire       : $histexpire "
     echo "histsize         : $histsize "
     echo "minage / maxage  : $minage /$maxage "
     echo "minalpha         : $minalpha "
     echo "maxexpired       : $maxexpired "
     echo "minother         : $minother "
     echo "minlen           : $minlen "
     echo "mindiff          : $mindiff "
     echo "maxrepeats       : $maxrepeats "

     # Give the user a chance of a nice break
     read answer </dev/tty
     if [ "$answer" = "q" ]
     then break
          fi
     done

# If lastlogin is not set: we did not find the user at all.
if [ ! "$lastlogin" ]
then echo "User $save_user not found"
     fi

# End script get_user
```

OUTPUT EXAMPLES

The first example is for adding users from one server to the database. Command:

```
     gather_aix server1.dom1.company
```

```
Mon Jun 11 20:35:39 DFT 2001 : begin script
********************************************************************
*
* Only data of server server1.dom1.company will be processed and
* added to the file users.aix
*
********************************************************************
Password of ftp user was found, continuing ...

Mon Jun 11 20:35:43 DFT 2001 : Processing server server1.dom1.company
Processing user root on server server1.dom1.company
Processing user daemon on server server1.dom1.company
.
.
.
Processing user user14 on server server1.dom1.company


Mon Jun 11 21:40:28  DFT 2001 : Executing users_to_remove
Output of users_to_remove can be found in /home/work_dir/aix_user/
remove.users


Mon Jun 11 21:41:35  DFT 2001 : Executing users_alert
Output of users_alert can be found in /home/work_dir/aix_user/
alerts.users

Mon Jun 11 21:41:47 DFT 2001 : end script
```

The second example shows getting output for one user (on one server). Command:

```
     get_user user11
```

```
Info of server   : server2.dom1.company
user             : user11
user id          : 246
primary group    : system
groups           : system staff bin sys adm mail cron usr shutdown dba
home directory   : /home/user11
user information : Userinfo for this user
login            : true
rlogin           : true
last login       : more than 1 year ago
su               : true
sugroups         : ALL
histexpire       :   0
histsize         :   0
minage / maxage  :   0 /  0
minalpha         :   0
maxexpired       :  13
minother         :   0
minlen           :   0
mindiff          :   0
maxrepeats       :   8
```

The third example is remove.users (generated by script users_to_remove).

```
Users who never logged in on all servers:
Userid       Server                   Login date           login  rlogin
adm          server1.dom1.company      more than 1 year ago  true   true
adm          server3.dom1.company       Never logged on       true   true
adm          server8.dom2.company       Never logged on       true   true
.
.
.
user99       server3.dom5.company       more than 1 year ago  true
false


Users who can partly revoked ( not on all servers) :
Userid       Server                     Login date          login  rlogin
amf          server2.dom1.company       Never logged on       true   true
amf          server6.dom1.company       more than 1 year ago  true   true
.
.
.
user15       server4.dom1.company       Never logged on       true   true


Users of which account is locked :
Userid       Server                     Login date          login  rlogin
user8        server4.dom1.company       Never logged on       true   true
user90       server4.dom1.company       Never logged on       true   true
user98       server4.dom1.company       Never logged on       true   true
```

The fourth example is alerts.users (generated by script user_alert).

```
user33 server3.dom5.company alert
          histexpire = 0
          histsize = 0
          minage = 0
          maxage = 0
          minalpha = 0
          minother = 0
          minlen = 0
          mindiff = 0
remote server5.dom5.company alert :
          histexpire = 0
          histsize = 0
sys server5.dom5.company alert :
          histexpire = 0
```

Teun Post
Unix/Oracle Specialist
Schuitema NV (The Netherlands)

SSA for beginners and beyond

Over the years I've accumulated a large amount of notes on dealing with SSA disk subsystems in an AIX-RS/6000 environment. I thought it would be helpful to organize them into an article that served as both a quick reference for the more advanced SSA administrator and an entry point to learning for the beginner.

OVERVIEW

SSA (Serial Storage Architecture) is basically IBM's answer to 'new and improved' SCSI. The disks themselves are still SCSI disk drives under the covers. Enhancements come into play with the loop architecture and full-duplex paths. SSA can send and receive data simultaneously and at full speed, where SCSI sends and receives on the same path. The disks themselves also are 'wrapped' with SSA specific electronics. The 16-address limitation of SCSI is also overcome with SSA disks. SSA disks are hot-pluggable and do not require special addressing like SCSI disks. Ease of use or introduction to your environment is further augmented by the fact that SSA is fully SCSI-2 compatible. No modification of applications is required.

THE LOOP

An SSA loop is the heart of the architecture. It will contain 4 to 48 disks, and at least two adapter ports. In the most basic loop, which I'll use for illustration purposes, 16 disks are attached to one pair of ports on an SSA Adapter. Each adapter contains two pairs of ports and is capable of supporting two separate loops – see Figure 1.

In this simple example SSA loop, we have 16 disks numbered 1 to 16 (when you look at an SSA drawer the disk bays in the front are actually labeled 1 to 8 on the front and 9 to 16 on the back). This number is etched into the disk bay 'cage'. The number 1 disk is cabled to port A1 on the adapter, and number 16 is cabled to A2, thus forming a loop. Because drawers are actually divided into quadrants, there are implicit connections between disks 4 and 5, disks 8 and 9, and disks 12 and 13.

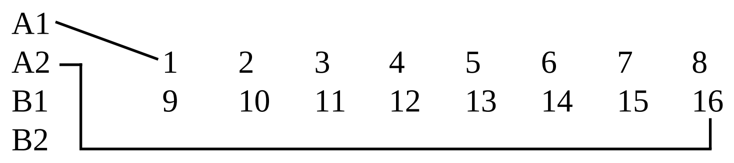

Figure 1: Simple SSA loop

This implicit connection is part of the drawer electronics. Each initiator (SSA adapter port) can reach any disk in the loop on two paths (there are two wires in the cable attaching the adapter to the disks). If a disk fails and the loop is 'broken', the loop will automatically reconfigure itself and reach its destination node (disk) via an alternative path. If you need to remove a disk from the configuration for a period of time, a blank 'dummy' disk is inserted in its place to maintain the loop continuity.

DISKS AND DISK DRAWERS

There are two styles of disk drawer – the original 7133-010 or 020 disk drawer and the 7133-D40. The 010/020 drawers had 2.2GB 4.5GB, and 9.1GB disks manufactured for them. The newer D40 had 4.5GB, 9.1GB, 18.2GB, and 36GB disks manufactured for it. As of this writing, the only drawer you can order new is the D40, with 9.1GB, 18.2GB, or 36GB disks. The older drawer and its accompanying disks, as well as the 4.5GB disk for the D40, are no longer orderable. All of the older equipment is very easily obtained on the used market. Disks designed for the D40 ('advanced' disks) will not work in the older 010/020 drawers, and *visa versa*.

Disk placement is done in quadrants. In our simple loop example, if you only had four disks to place in the loop, you would do so based on minimum per quadrant placing. The quadrants are 1-4, 5-8, 9-12, and 13-16. You must have at least two disks per quadrant. You could not therefore place them in disk bays 1, 5, 10, and 14. You could however have them in 1 and 2 and 13 and 14, for example. The remaining open bays would have blanks inserted.

CABLES

There are two different SSA cables that come in 1, 2.5, 5, 10, and 25-metre lengths.

There is a unique ‘D’ style connect on each end. A special tool comes with most SSA purchases to assist in tightening down the cables. The older black SSA cables are compatible with the older 7133-010/020 drawers/disks, while the new blue cables are designed to be used with the newer 7133-D40 drawers/disks. The ‘D’ connectors are exactly the same, but the cables themselves are capable of higher (double) speeds.

You can use the blue cables with the older equipment, but no performance difference will be seen. Likewise you can use the Black cables with the newer disks, but performance degradation will most likely occur.

FIBRE OPTIC EXTENDERS

The maximum SSA cable length is 25 metres. Fibre optic extenders are available that will allow host connections from 2km to 10km away, depending on whether you are using single or multi-mode fibre. There are a number of rules surrounding fibre use. There is an excellent outline of them at http://www.hursley.ibm.com/~ssa/docs/fibreusage.html

LOOP SPEEDS

Loop speeds for the older style configuration have a theoretical maximum of 80MB/sec, while the new style configurations have a theoretical maximum of 160MB/sec.

ADVANCED LOOP CONFIGURATIONS

There are a myriad of supported loop configurations, diagramming of which is outside the scope of this article. Depending on the adapter and RAID configuration, you can have from 1 to 8 adapters participating in one loop, for both parallel access and High Availability. In the former case, you need software that supports parallel access to the logical volumes on the disk, like Oracle Parallel Server or GPFS. The

Redbook *Understanding SSA Subsystems in Your Environment*, SG24-5750, has superb diagrams and explanations of possible loop configurations. It is also available online at http://www.redbooks.ibm.com/redbooks/SG245750.html for the HTML version or http://www.redbooks.ibm.com/pubs/pdfs/redbooks/sg245750.pdf for the PDF format.

GLOBAL SSA LOOP CABLING RULES

While there are a number of adapter-specific cabling rules, there are some global rules as well:

- A minimum loop passes through one pair of connectors on an adapter card: either A1 and A2 or B1 and B2. There is no performance difference between the two.
- If two adapters are present in the loop, it may use one set of connectors on each. You cannot use both sets of connectors on one adapter in the same loop.
- You can have a maximum of three contiguous dummy disks in a loop.
- The maximum cable length between adapter and disks is 25 metres with copper cables, 2.3km with single mode fibre, and 10.5km with multi-mode fibre.
- In loops containing JBOD disks, you may mix different drive sizes. If RAID-5 is used, drive geometry must be the same.

FAST-WRITE CACHE

Fast-write cache is available for the 6215 (4MB), 6219 (4MB), and 6225 (32MB) adapters. The cache is an optional/chargeable feature. My own experience with fast-write cache is with the 6225. Just to give you a very rudimentary example of its benefits: we had an approximately 25GB Oracle database, whose import of a 5GB dump file took 42 hours prior to using the fast-write cache. The database was in a test/development environment and was spread over a relatively small number of disks. By adding the fast-write cache, and changing nothing else, the import was reduced to 7 hours. For the 6225 adapters, the fast-write cache is a relatively low-cost portion of the overall adapter cost.

ADAPTER-SPECIFIC RULES

Figure 2 is an excerpt from IBM documentation. It deals only with RS/6000 host adapters.

Adapter	*Characteristics*	*RAID Support*	*AIX Support*
6214(4-D)	Micro Channel, non-RAID or JBOD. There can be two in one loop.	None	3.2.5, 4.1.4, 4.1.5, 4.2.1, 4.3.*x*
6216(4-G)	Micro Channel, non-RAID. You can have up to eight in a loop. If a 6214 is mixed into the loop, You may only have one of each.	None	3.2.5, 4.1.4, 4.1.5, 4.2.1, 4.3.*x*
6217(4-I)	Micro Channel RAID-5 or non-RAID. There may only be one per loop. You can mix RAID and non-RAID disks in the same loop.	None	4.1.5, 4.2.1, 4.3.*x*
6219(4-M)	Similar to 6217, but you can have two adapters in a loop with RAID and 8 non-RAID. There is a fast-write cache option available for this adapter.	RAID 5	4.1.5,4.2.1, 4.3.x
6218(4-J)	Same functions as 6217, but for PCI hosts.	None	4.1.5, 4.2.1, 4.3.*x*
6215(4-N)	Same functions as 6219, but for PCI hosts.	RAID 5	4.1.5,4.2.1, 4.3.x
6225(4-P)	Advanced Adapter. Higher loop speeds (with advanced cables and disks). May have two adapters in loop, RAIDed, and 8 non-RAID.	RAID 0,5	4.2.1, 4.3.*x*
6230(4-P)	Same as 6225, but increased cache capability	RAID 0,5	4.2.1, 4.3.*x*

Figure 2: RS/6000 host adapters

HOT SPARES

When RAID is in use, you may have one or more hot spares designated. If a drive in the RAID array fails, the hot spare will automatically take over. If you are using 6217 or 6218 adapters, the hot spare may reside outside of the loop. For all other RAID supported adapters, the hot spare must reside in the loop.

OTHER HOSTS

It is worth a mention that SSA configurations are not limited to RS/6000 hosts. You can also attach them to Sun, HP, and Intel (NT) hosts. There is separate documentation available for these configurations online at the IBM SSA site, and in hardcopy.

STORAGE AREA NETWORKS

There are some relatively new offerings that allow you to use SSA configurations in a SAN environment. A ‘SLIC’ router can control up to 64 disks. Software is provided to control the disk RAID configuration and target-lun masking for SCSI-based systems.

A visit to http://www.vicom.com will give you loads of information on this hardware.

SERIAL STORAGE EXPERT (STORX)

StorX is optional software available from IBM, which allows a visual graphical interface to managing SSA configurations. In my own experience it is more useful as a visual aid for planning than it is in actual configuration. More information on this product can be found at http://www.storage.ibm.com/software/storwatch/storx/index.htm.

MANAGING DISKS IN AIX

SMIT screens are available to enter a wealth of SSA commands, many of which are available directly via a command line. There are commands to list SSA disks on a system, list disks attached to specific adapters, list adapters assigned to specific disks, RAID management, a utility to identify a disk by lighting an indicator on it, and many more. The SMIT panels are well documented and easy to use.

MICROCODE

Microcode is required for both disks and adapters. When you install later versions of AIX, basic SSA microcode will be present. Occasionally for enhancements or fixes, microcode updates are released. There is an excellent matrix of microcode support by adapter, disk, and AIX release, as well as the ability to download it, from http://www.hursley.ibm.com/~ssa/rs6k/index.html. The software is an AIX .bff installable file(s). Just installing the programs/code, does not update the microcode on the disks and/or adapters. You must either run *cfgmgr* (it's good to run it twice to ensure that the VPD gets updated), re-boot, or use DIAG panels for SSA management to load the code. Also note that you cannot 'un-apply' non-committed filesets once the microcode has been loaded. This will not be effective in removing the microcode. There is a more in-depth discussion of this in the documentation.

While it is theoretically possible to update microcode concurrently, with the disks and adapters active, IBM recommends (strongly) that all activity to the disks be quiesced. My own experiences confirm this.

FREQUENTLY ASKED QUESTIONS

There is a comprehensive list of FAQs at http://www.hursley.ibm.com/~ssa/faq.html.

It is too large to replicate here, but a worthwhile read. It contains hints and tips on adapter placement for performance, what to do (or to ignore) about errors in the AIX error log, and so forth.

FINAL COMMENT

As you can see, SSA subsystems range from the relatively simple to very complex. There has been some debate as to who is responsible for what components, because software, microcode, and hardware expertise are required to effectively design and administer all of the components. IBM recommends, as do I based on personal experience, that you make it a joint effort between yourself as the system administrator and your IBM hardware CE.

REFERENCES

IBM Redbook, *Understanding SSA Subsystems in Your Environment*, SG24-5750-00. (Note: this Redbook is highly recommended if you are new to SSA environments.)

All but 6225 and 6230:

- IBM *The Problem Solving Guide and Reference*, SC23-2204.
- IBM *Diagnostic Information for Micro Channel Bus Systems*, SA23-2765.
- IBM *Diagnostic Information for Multiple Bus Systems*, SA38-0509.
- IBM *SSA Adapters Users Guide and Information* , SA33-3272-02.

6225 and 6230:

- IBM *Advanced Serial Raid Adapters*, *Users Guide and Maintenance*, SA33-3285-02.
- IBM *Problem Solving Guide and Reference*, SC23-2204.
- IBM *Diagnostic Information for Multiple Bus Systems*, SA38-0509.

David Miller
Database Architect
Baystate Health Systems (USA)

Quick reference: Solaris to AIX – part 2

This month we conclude the quick reference, contrasting AIX Version 4.3.3 and Sun Solaris 8 operating systems.

FILE SYSTEM MANAGEMENT

Figure 1 shows tasks that are employed when performing file system management in AIX and Solaris.

Tasks	*AIX Version 4.3.3*	*Solaris 8*
Run multiple tasks in a GUI environment	smit or wsm	N/A
Format a disk	N/A - Automatically handled	format
Check a file system	fsck	fsck
Mount a file system	mount	mount
Display available file-system space	df	df
Partition a disk	N/A - Automatically handled	format
List a volume's table of contents	lchangelv	prtvtoc
Add a file system	crfs	newfs or mkfs
Unmount a file system	umount	umount
Back up file systems/ files/directories	backup	ufsdump
Restore file systems/ files/directories	restore	ufsrestore

Figure 1a: File system management

Change a file system	chfs	tunefs
Remove a file system	rmfs	N/A
Display a file system	lsfs	/etc/vfstab

Figure 1a: File system management

VIRTUAL DISK MANAGEMENT

Figure 2 is a list of tasks that are used when implementing virtual disk management in AIX and Solaris:

Tasks	*AIX Version 4.3.3*	*Solaris 8*
Run multiple tasks in a GUI environment	smitty chjfs or wsm	metatool
Expand file system	chfs or smitty chjfs	growfs
Delete metadevice	N/A	metaclear
Configure metadevice	N/A	metainit
Modify metadevice	N/A	metaparam
Rename metadevice	N/A	metarename
Display status of metadevice	N/A	metastat

Figure 2: Virtual disk management

LOGICAL VOLUME MANAGEMENT

Figure 3 is a list of tasks that are used when performing logical volume management in AIX and Solaris. The information in this table includes Solaris 8 using the Veritas Volume Manager (VxVM). IBM includes its Logical Volume Manager (LVM) in AIX Version 4.3.3.

Tasks	*AIX Version 4.3.3*	*Solaris 8*
Storage Structure Terminology	A disk is composed of physical partitions. A physical volume is the same thing as a disk. A volume group is composed of physical volumes. A volume group is divided into logical volumes. A file system is placed onto a logical volume. A file system is placed onto a partition.	A disk is composed of partitions/slices. A subdisk (somewhat similar to AIX physical partition) is composed of partitions/slices. A plex (similar to AIX logical partition) is composed of subdisks. A volume (similar to AIX logical volume) is composed of plexes. A VM disk is composed of subdisks. A disk group (similar to AIX volume group) is composed of VM disks.
Run multiple tasks in a GUI environment	smit or wsm	vxva
Move logical volume to another physical volume	migratepv	vxassist move
Create logical volume	mklv	vxassist make
Extend logical volume	extendlv	vxassist growto
Remove logical volume	rmlv	vxedit rm
Set up sysboot information on VM disk	N/A	vxbootsetup
Manage VM disks	N/A	vxdisk
Create volume group	mkvg	vxdg init

Figure 3a: Logical volume management

Remove disk from volume group	reducevg	vxdg rmdisk
Add disks under volume manager	extendvg	vxdiskadd
Administer disks	reducevg or extendvg	vxdiskadm
Set up disks	extendvg	vxdisksetup
Change logical volume settings	chlv	vxedit set
Create configuration records for storage structures	mkvg or mklv	vxmake
Manage plexes or volume groups	chvg or mkvg	vxplex
Display volume group	lsvg	vxprint
Change size of volume	lchangelv	vxresize
Manage subdisk or physical volume	chpv	vxsd
Display statistics for storage structures	Choose one of the following: lspv lsvg lslv	vxstat
Manage volume	Choose one of the following: chlv mklv rmlv	vxvol
Back up operating system	mksysb (to tape or file) or mkcd (CD-ROM)	Solstice Backup: nwadmin

Figure 3b: Logical volume management

Restore operating system	mksysb (to tape or file) or mkcd (CD-ROM)	Choose one Solstice Backup: nwadmin nwrecover

Figure 3c: Logical volume management

TROUBLESHOOTING AND ADDITIONAL LOCATION INFORMATION

Figure 4 includes troubleshooting and additional location information in AIX and Solaris.

Tasks/Locations	*AIX Version 4.3.3*	*Solaris 8*
Change a host name	chdev -l inet0 -a hostname=host name	Minimum change required for the followings files: /etc/nodename /etc/hosts /etc/hostname.* /etc/net/*/hosts
List of well-known networking services and port numbers	/etc/services	/etc/services
List of well-known protocols	/etc/protocols	/etc/protocols
Provide interface-level packet tracing for Internet protocols	iptrace	snoop
Display network status	netstat	netstat
Display NFS and RPC statistics	nfsstat	nfsstat
Display statistics on network I/O and network CPU usage	netpmon	N/A

Figure 4a: Troubleshooting and other location information

Display a snapshot of virtual memory	svmon	N/A
Display virtual memory statistics	vmstat	vmstat
Display I/O statistics	iostat or filemon	iostat
Report system activity	sar	sar
Display simple and complex lock contention information	lockstat	N/A
Report CPU usage	tprof	N/A
Simulate a system with different memory sizes for performance testing	rmss	N/A
Display system error log	errpt -a	dmesg
Display/Set dump device	sysdumpdev	N/A
Display paging/swapping space	lsps -a	swap -l
Specify users who have access to cron. (Every user has access to cron if the access file does not exist.)	/var/adm/cron/cron.allow	/etc/cron.d/cron.allow
Specify users who have no access to cron	/var/adm/cron/cron.deny	/etc/cron.d/cron.deny
Specify remote users and hosts that can execute commands on the local host	/etc/hosts.equiv	/etc/hosts.equiv

Figure 4b: Troubleshooting and other location information

Information about mounting local and remote file systems	/etc/filesystems	/etc/vfstab
Display information about the current mounted file systems	mount	mount orpg /etc/mnttab
Default superuser log	/var/adm/sulog	/var/adm/sulog
Configure syslogd logging	/etc/syslog.conf	/etc/syslog.conf
Display physical RAM	bootinfo -r	prtconf

Figure 4c: Troubleshooting and other location information

ADDITIONAL SOLARIS COMMANDS IN AIX 5L

To help Solaris-specialized system administrators adapt more easily to AIX in their heterogeneous environments, additional Solaris commands and tools are available in AIX 5L for common system administrator tasks. For example:

- The **prtconf** command conveniently displays system information such as memory, hardware architecture, network configuration, paging space, and system device availability.
- The **rmtcpip** command removes the network interface configurations such as host name, IP address, subnet mask, gateway address, and name server information. It restores your system network to a pristine state.

Werner Klauser
Klauser Informatik (Switzerland)

Understanding the wc command

The **wc** or wordcount command is a versatile tool that can assist in many different tasks. **wc** counts the number of lines, words, or bytes in a file.

Suppose you had a raw data file with one line per entry and you want to know how many entries you had. A tool that displays the number of lines in a file would help.

Or maybe you are a technical writer and need to determine the number of words in a document to get translation cost estimates. A tool that counts words in a file would do that.

Perhaps you want to determine the total number of bytes in a group of files to determine whether they would fit onto a diskette. A tool that counts and displays the number of bytes would provide an answer.

WC COMMAND BASICS

The **wc** command takes as input the name of the file or files for which you want a count of the lines, words, and/or bytes. The default behaviour is to display all three counts on one line followed by the name of the file. To display only one of the three values presentable by **wc**, you can use one of the available flags.

The basic syntax of the **wc** command is:

```
    wc flags filespec
```

where:

- *flags* is an optional flag or flags used to enhance the **wc** operation.
- *filespec* is the file or files on which the **wc** operation is to act.

FLAGS FOR THE WC COMMAND

The following flags help to enhance the usefulness of the **wc** command:

- *-l* – counts lines, which is actually a count of newline characters.

- *-w* – counts words, which are defined as strings that are separated by spaces, tabs, or newline characters.
- *-c* – counts the total number of bytes in the input file specification.

USING THE WC COMMAND

Suppose you had a file called mychap.txt and wanted to know how many lines, words, and bytes were in the file. If you were to enter:

```
wc mychap.txt
```

you may get the following result:

```
236    860    5597 mychap.txt
```

This would tell you there are 236 lines, 860 words, and 5597 bytes in file mychap.txt. If you wanted to show only the total number of lines in the file, enter:

```
wc -l mychap.txt
```

and you would get the following response:

```
236 mychap.txt
```

The syntax for displaying only the word or byte counts would be similar, but you would use the *-w* or *-c* flags.

You can also combine the flags to show multiple counts for a file. Suppose you were interested in knowing the line count and word count for a documentation file called chapter1.doc, without concern for the number of bytes in the file. Enter:

```
wc -lw chapter1.doc
```

which may yield the following response:

```
122    752 chapter1.doc
```

This tells you chapter1.doc contains 122 lines and 752 words. No byte count is displayed.

Note that it makes no difference whether you entered *-lw* or *-wl* on the **wc** command, you will get the same order in the results: line count, then word count. The **wc** command would just see that you want both counts and would not try to place the results in the order you specified, but rather the default order of the **wc** command output – lines, words, then bytes.

MULTIPLE FILES

The **wc** command can also work with multiple files. For example, suppose you had five chapter files for a book with the following names:

```
chapter1.doc
chapter2.doc
chapter3.doc
chapter4.doc
chapter5.doc
```

If you wanted to get a total number of lines and words for all the chapters, you could enter:

```
wc -lw chapter*.doc
```

which may give the following results:

```
122     752 chapter1.doc
236     860 chapter2.doc
 67     302 chapter3.doc
259    1163 chapter4.doc
153     838 chapter5.doc
837    3915 total
```

In this case, the **wc** command displays lines and words for each file, followed by the file name. The last line of the output contains the total number of lines (837) and words (3915) for the group.

PIPING INTO WC

You can pipe the output of a command into the **wc** command to get a count. For example, if you were to enter:

```
cat chapter1.doc | wc -l
```

the result would be:

```
122
```

This is the total number of lines in the file. The **cat** command types the file through the pipe and into the **wc** command, which counts the lines. Note that when the input to **wc** comes from standard input, the file name is not displayed. This would be useful if you wanted the output to be just the numeric results, perhaps as input into a table, or another filter or process.

If you were to enter:

```
cat chapter*.doc | wc -l
```

the result would be:

```
837
```

This gives you the total number of lines in all the chapter files without displaying individual file results or names.

SOME PRACTICAL EXAMPLES

Let’s see how the information presented in this article can be used to solve the problems defined in the three scenarios described in the introduction:

1 A user wants to count the number of aliases he has set up. He uses the alias command and then pipes the output into the **wc** command and specifies the *-l* flag to count and display only lines:

```
alias | wc -l
```

The response he gets, say 18, indicates that he has set up 18 aliases on the system, the total number of lines in the alias file.

2 A user wants to count the number of words contained in the prologue of one of his modules. The first 22 lines of module example1.c are the prologue, so he uses the head command to capture the first 22 lines and then pipes the output into the **wc** command and specifies the *-w* flag to count and display only words:

```
head -22 example1.c | wc -w
```

The response he gets, say 67, represents the total number of words in the first 22 lines of the file example1.c.

3 A user wants to count the total number of bytes in a group of his documentation files to see whether they will all fit onto a high density diskette, which holds about 1.4MB of data. The files all have the format DOCDATA*n*.TXT where n is a number from 1 to 8. He uses the **cat** command and then pipes the output into the **wc** command and specifies the -*c* flag to count and display only bytes:

```
cat DOCDATA*.TXT | wc -c
```

The response he gets, say 1800210 (1.8MB), indicates the total number of bytes in the files, and that there is too much data to fit onto a single diskette.

SOME EXERCISES

Here are some exercises to test your knowledge of the **wc** command.

Exercise 1

1 Find a text file such as a product README file and copy it to a directory to which you have access. The remaining steps of this exercise will refer to the file as README.

2 Enter the following command:

```
wc README
```

Note the response is three figures – the line count, the word count, and the byte count, followed by the name of the file, README.

3 Enter the following command:

```
wc -l README
```

Note the response is the same number of lines as displayed in Step 2, and the file name. No other figure is displayed.

4 Enter the following command:

```
wc -w README
```

Note the response is the same number of words as displayed in Step 2, and the file name.

5 Enter the following command:

```
wc -c README
```

Note the response is the same number of bytes as displayed in Step 2, and the file name.

6 Enter the following command:

```
wc -lw README
```

Note the response is the same number of lines and words as displayed in Step 2, but no byte count, followed by the file name.

You have just exercised the **wc** command by itself, and with the various flags to display all or parts of the available output of the **wc** command.

Exercise 2

1 Suppose you wanted to determine how many users were currently logged into the system. How could you use the **wc** command to help you?

Issue the following command:

```
who | wc -l
```

Note the response is the number of users. If you were to simply enter **who** you would get a table with login IDs, dates, and other login information. By piping the output of the **who** command into the **wc** command, and specifying the *-l* flag, you have asked **wc** just to give you the line count of the output of the **who** command, resulting in the display of the number of users logged in.

2 Change to a directory that has about ten or twenty files in it and issue the following command:

```
ls | wc -w
```

What do the results tell you? The number that displays is the number of files contained in the current directory expressed as the number of ‘words’.

Now issue the following command:

```
ls -l | wc -w
```

What does this new, larger number mean? Now, since you have asked for the 'long list' to be piped into **wc**, the number of words displayed will include all the additional information the *ls* command provides such as the permissions, owner, group, and date. You may issue the *ls -l* command without piping into **wc** to verify what the **wc** would be processing. This additional information may not be useful to you. Therefore, it is important to know what your input to the **wc** command is before you rely on the data.

3 Using the same directory as in Step 2, enter the following command:

```
cat * | wc -c
```

What do the results tell you? The number that displays is the total number of bytes in all the files in the current directory. Would they all fit onto one 1.4MB diskette?

You have just exercised piping data into the **wc** command to display counts of lines, words, and bytes without displaying extraneous data.

David Chakmakian
Programmer (USA)

AIX news

IBM will make 64-bit Java technology available on AIX, claiming to leapfrog Solaris and HP-UX.

Existing Java applications can be moved to the 64-bit environment without recompiling or recoding.

Partner Micromass has already begun selling hardware and software bundles for managing, analysing, and storing information gathered by Micromass devices.

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

* * *

IBM has announced enhancements to its Tivoli Storage Management products, including the addition of LAN-free client data transfer for AIX, Solaris, NT, and Windows 2000.

The software exploits the SAN path by enabling the Tivoli Storage Manager client to back up and restore data directly to and from SAN-attached storage.

There's additional support for LAN-free client data transfer directly from the client to a SAN-attached storage device. Also, the Tivoli Storage Manager HP-UX server can now participate in tape library sharing and the Tivoli SANergy exploitation enables LAN-free client data transfer to sequential access storage pools on disk.

The 3494 library includes support for LAN-free client data transfer and for dynamic drive sharing.

There's also support for SCSI tape failover in a Windows cluster configuration of Tivoli Storage Manager servers.

Other enhancements include improved performance when writing to 3590 tape devices or devices which emulate 3590s on OS/390 Version 2.10 or z/OS systems.

Finally, the new Tivoli Data Protection for NDMP supports Network Appliance NAS file servers.

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

* * *

IBM has announced Version 2.0 of its Document Connect for both Lotus Notes and Domino for Multiplatforms, with new support for AIX for the Domino version.

Among the new bits are the ability to create one logical document, based on multiple document templates, each with its own formatting characteristics, and a new step back function, letting users make corrections before proceeding with the creation of a document.

There's also an extended editor applet for adding or modifying free text from a Web browser, new actions are available for Web browser users, and there is support for framesets for template developers.

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