36

AIX update

October 1998

In this issue

AIX Update

Published by

Xephon
27-35 London Road
Newbury
Berkshire RG14 1JL
England
Telephone: 01635 550955
From USA: 01144 1635 33823
E-mail: HarryLewis@compuserve.com

North American office

Xephon/QNA
1301 West Highway 407, Suite 201-405
Lewisville, TX 75077
USA
Telephone: 940 455 7050

Contributions

If you have anything original to say about AIX, or any interesting experience to recount, why not spend an hour or two putting it on paper? The article need not be very long – two or three paragraphs could be sufficient. Not only will you actively be helping the free exchange of information, which benefits all AIX users, but you will also gain professional recognition for your expertise and that of your colleagues, as well as being paid a publication fee – Xephon pays at the rate of £170 ($250) per 1000 words for original material published in AIX Update.

To find out more about contributing an article, see *Notes for contributors* on Xephon's Web site, where you can download *Notes for contributors* in either text form or as an Adobe Acrobat file.

Editor

Harold Lewis

Subscriptions and back-issues

A year's subscription to *AIX Update*, comprising twelve monthly issues, costs £175.00 in the UK; $265.00 in the USA and Canada; £181.00 in Europe; £187.00 in Australasia and Japan; and £185.50 elsewhere. In all cases the price includes postage. Individual issues, starting with the November 1995 issue, are available separately to subscribers for £15.00 ($22.50) each including postage.

AIX Update on-line

Code from *AIX Update* is available from Xephon's Web page at www.xephon.com (you'll need the user-id shown on your address label to access it).

Migrating a mail system to AIX

INTRODUCTION

A few years ago, our management decided to migrate the academic mainframe system (VM/CMS) to Unix. The migration process included many phases and took many aspects into consideration. One major part of the process was to migrate the mailer system – electronic mail was one of the most heavily used systems on the VM/CMS mainframe. As the users' stored e-mail files were valuable and needed to be available on the new platform, we needed to build a utility to migrate them. Users should be able to use the default mailer interface on the new platform to perform various tasks on messages created using the old system. Those tasks include read, reply, forward, delete, save in another file, etc.

On the VM/CMS system, we used RiceMail, which uses *NOTEBOOK* files to store e-mail messages and *NAMES* files to store addresses, as our mailer system interface. On the Unix system, we set Pine, which uses folders to store e-mail messages and *.addressbook* files to store addresses, as the default mailer system interface.

This article essentially shows you how to handle this type of migration problem, using the e-mail migration exercise as an example. The approach described is suitable for migrating many types of application and data, and you may also find that you can adapt the utility itself for your own use, as the accompanying text gives you enough information about the utility to make it a suitable framework for building your own.

The article uses the example of handling *NOTEBOOK* and *NAMES* files and converting them to Unix folders and *.addressbook* files. However, the procedure to analyse the format of these files is applicable to migration to and from any platform, not just from a VM/CMS mail system to Unix. This is because of the standardization of mailer systems used on numerous platforms as a result of a number of Requests For Comments (RFCs). The division of electronic mail messages into a header and body, and the syntax and order of header lines, are all defined in RFC822, titled *Standard for the Format of ARPA Internet Text Messages*. This RFC also describes the form of addresses. On the other hand, address files are dependent on the particular e-mail system.

This article focuses on analysing an electronic mail message, the format of the *NOTEBOOK* and *NAMES* files, the format of Unix folders and *.addressbook* files, and finally a simple utility to convert *NOTEBOOK* and *NAMES* files into folders and *.addressbook* files respectively.

AN ELECTRONIC MAIL MESSAGE

Typically, an electronic mail message consists of two parts: a header and a body. The header consists of all the lines of text from the first line until at least one blank line is encountered. The body is everything from the blank line to the end of the message. Simply, the body is the actual text of the message and the header is the information used by the mailing systems to deliver the electronic mail message. Our discussion concentrates on the header.

The header contains information about the return address of the sender, the electronic mail message propagation through the network, the sender's address, the subject, the recipient's address, the date of receiving the message, the message identification number, and brief description of the content of the message. It may also contain other information that is in most cases optional. Many header lines can appear in an electronic mail message; some are required, some are optional, and some may appear many times.

In an ideal case, the header should look like the following:

```
Return-Path: <originator-username@originator-nodeid>
Received: from originator-nodeid (originator mailer name and type) by
          destination-nodeid (destination mailer name and type)
          with BSMTP id xxxx; date and time of reception
Received: from originator-nodeid (originator mailer name and type) with
          BSMTP id yyyy; date and time of sending
From:     Someone <originator-username@originator-nodeid>
Subject:  Migration from VM to UNIX
To:       recpient-username@recpient-nodeid
Cc:       someone-username@his-nodeid
Date:     day, date, time and probably time zone of sending
```

```
Message-ID: <message id of the originating message>
MIME-Version: aa.bb
Content-Type: type of the message text; charset=character set by the
              originator mailer
X-Mailer: type of originating mailer system and probably version
```

Where *aa.bb*, *xxxx*, *yyyy*, and *zzzz* are a combination of alphanumeric characters.

The required header lines are '*Received:*', '*From:*', '*Date:*', and '*Message-ID:*'; others are optional. The '*Received:*' header line may appear many times as the message propagates through the network and mailing systems 'stamp' it. There are also some other types of header line, though you are unlikely to come across them.

NOTEBOOKS VERSUS UNIX MAIL FOLDER

Typically, the *NOTEBOOK* file follows the following format:

```
( separator line )
Header part of the first message
( at least one blank line )
Message text
( separator line ) *
Header part of the next message *
( at least one blank line ) *
Message text *
.
.
.
```

Note that parts marked with an asterisk ('*') are repeated (in order) up to the end of the file. The Unix mail folder follows the same format as the *NOTEBOOK* file, albeit with a slight difference:

```
( separator line )
Header part of the first message
( at least one blank line )
Message text
( at least one blank line ) *
( separator line ) *
Header part of the next message *
( at least one blank line ) *
Message text *
.
.
.
```

Again, lines marked with an asterisk are repeated, in order, up to the end of the file.

NOTEBOOK VERSUS UNIX MAIL FOLDER ANALYSIS

The most important issue here is how to identify and process the separator line. The first message should be preceded by a separator line. Each message is then separated from the previous one by a separator line. Also, there is at least one blank line separating the header from the message body.

For some reason, the old mailer system sometimes used 72 equal signs as the separator line, and at other times used 73. So we had to consider both cases. On the other hand, we noted that the Unix mail folder's separator line is preceded by at least a blank line (except for the first message) and has the following format:

```
From originator-electronic-mail-address ddd mmm xx aa:bb
time-zone yyyy
```

Where:

- *originator-electronic-mail-address* is the address of the originator in the form *userid@nodeid.*
- *ddd* is the day the message was received, from the set [Sat, Sun, Mon, Tue, Wed, Thu, Fri].
- *mmm* is the month the message was received, from the set [Jan, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct, Nov, Dec].
- *xx* is a two-digit number, from the set [01-31], representing the day of the month that the message was received.
- *aa:bb* is the time that the message was received.
- *time-zone* is the time zone of the receiving mailer system – if any.
- *yyyy* is a four-digit number representing the year the message was received.

NAMES VERSUS UNIX ADDRESS BOOK FILES

Typically, the *NAMES* file has the following format:

```
    :nick.nickname  :userid.userid  :node.nodeid   :name.actual name
```

NAMES uses tags to identify the values of each nickname entry. Some of these tags are required, others are optional.

The required tags are:

- *:nick.<the nick name>*
- *:userid.<the user-id associated with the nick name>*
- *:node.<the node-id of the user-id>*

The optional tags are:

- *:name.<the actual name of that person>*
- *:notebook.<the NOTEBOOK file used to store messages>*
- *:phone.<the phone number for that person>*
- *:FAX.<the fax number for that nick name>*
- *:DESCRIPTION.<some description for that nick name>*
- *:addr.<the post mail address for that nick name>*
- *:list.<a list of nicknames and/or e-mail addresses>*

The Unix *.addressbook* file conforms to the following format:

```
    (nickname) tab (actual name) tab (userid@nodeid)
```

Where:

- *(nickname)* is the person’s nickname (without parentheses).
- *(actual name)* is the person’s name (without parentheses).
- *(userid@nodeid)* is the person’s e-mail address (again, without parentheses).

NAMES VS. UNIX ADDRESS BOOK ANALYSIS

Among the various types of tag used in the *NAMES* file, *:nick.*, *:userid.*, and *:node.* are the most important. The optional *:name.* can be used, if present. Other *NAMES* tags are ignored.

The key element here is the *:nick.* tag. Every tag that follows *:nick.* belongs to it until another *:nick.* tag is encountered.

THE UTILITY

The utility assumes that the *NOTEBOOK* and *NAMES* files have already been moved to the new platform, and have names that conform to the naming convention *filename.NOTEBOOK* and *filename.NAMES*, respectively.

The utility is designed to process files in two separate steps: firstly, to convert *NOTEBOOK*s to Unix folders, and, secondly, to convert *NAMES* files to *.addressbook* files.

NOTEBOOKS TO UNIX FOLDERS

Following the analysis, we treated the first message as a special case. The first message is processed as follows:

1 The first line is assumed to be a separator line (but is nevertheless checked).

2 The separator line is converted.

3 The header is copied ‘as is’.

4 A blank line is inserted to separate the header from the body.

5 The body is copied ‘as is’.

Messages other than the first one are processed as follows:

1 Check that the line is a separator line.

2 Create two blank lines.

3 Carried out steps 2, 3, 4, and 5 above.

4 Repeat this process till the end of the input file is reached.

As the separator line doesn’t interfere with the mailer system interface, we used ‘*From oprnad@saupm00 Sat Jan 01 00:00 GMT 1995’* as the separator in the converted folders.

NAMES TO UNIX ADDRESSBOOK

We first reformat the *NAMES* file so that every line starts with the *:nick.* tag followed by its associate tags. We then strip the values of the *:nick., :userid., :node.*, and *:name.* tags and ignore the rest. Having these values, we can construct the *.addressbook*.

The tags *:nick., :userid.*, and *:node.* in the *NAMES* file should always be in this order; other tags may appear in any order.

Some lines may contain the *:list.* tag; in such cases the associated *:nick.* tag may not contain *:userid.* and *:node.* tags. Such a combination would produce unpleasant results.

THE SCRIPT

We called the utility script **nncsys** which stands for *Notebook and Names Conversion System*. The script assumes that the user is in the directory where the files to be converted reside. Otherwise, the script returns a message that the ‘file does not exist’.

NNCSYS

```
#! /bin/csh
#
# C Shell script for Notebook to Addressbook migration
#
set stop = 0                # reset loop termination flag
set executable
set continue = n

while ($stop == 0)          # loop until done
clear
  cat << MAIN_MENU          # display menu

            -----------------------------------------
            | Notebook and Names file migration tool |
            -----------------------------------------
```

```
        1  -> Usage Notes                   username: $LOGNAME
        2  -> List of file(s)               Date:   `date '+%a %b %d %Y'`
        3  -> Convert a VM notebook         Time:   `date '+%H:%M'`
        4  -> Convert a VM names file
        5  -> Go to PINE
        6  -> Exit


MAIN_MENU

  echo -n 'Enter your choice? '           # prompt
  set reply = $<                          # read response
  switch ($reply)                         # process response

#
# Case 1 : Usage Note
#
case "1":
clear
echo ' Notebook and Names Conversion SYStem (NNCSYS) Usage Notes'
echo ' ========================================================='
echo ' - You have to be in the directory where the files to be'
echo '   converted reside. If you are not, please exit using'
echo '   option 6 and change directory.'
echo ' - Option 1 shows this message.'
echo ' - Option 2 shows the list of files in the current directory.'
echo ' - Option 3 converts a VM Notebook to a Pine folder. The new'
echo '   folder is in the mail subdirectory of your home directory'
echo '   as (your-old-file-name)-notebook (without parentheses).'
echo ' - Option 4 converts VM NAMES files to a Pine addressbook'
echo '   file in your home directory under the name .addressbook.'
echo '   If you already have an addressbook, you are prompted to'
echo '   save the old one as .addressbook.bak. You may choose to'
echo '   merge the old addressbook with the new one. You can also'
echo '   choose to delete.addressbook.bak.'
echo ' - Option 5 uses Pine for checking the converted files.'
echo ' - Option 6 exits the utility.'
echo " "
echo Press ENTER to return to the main menu ....
echo " "
echo " "
while ($< != "")
end

breaksw

#
# Case 2 : List of files in the current directory
#
case "2":
```

```
clear
ls -al | more
echo " "
echo Press ENTER to return to the main menu ....
echo " "
echo " "
while ($< != "")
end

breaksw

#
# Case 3 : Notebook to Pine folder conversion
#
case "3":
clear
echo " "
echo " "
echo -n 'Enter the name of the notebook (without the .notebook
extension): '
set notebookname = $<
set inputfile = ${notebookname}.notebook
set outputfile = $HOME/mail/${notebookname}-notebook

echo ' '
echo 'Notebook file to be converted is: ' $inputfile
echo ' '
echo 'Output Pine folder is: ' $outputfile
echo ' '

# Uncompress if it is compressed
if ( -f ${notebookname}.notebook.Z) then
     uncompress ${notebookname}.notebook.Z
endif

# Does the target Notebook exist ?
if !( -f $inputfile) then
# NO ! Send the user a message
    echo 'File ' $inputfile ' does not exist'
else
# Yes :) OK start
  echo 'Conversion Started......'
  awk 'BEGIN {first = 1; \
              CNTR = 1} \
 {line72="========================================================================"
\
 line73="========================================================================="
\
   COLON=":" \
   if (($1 == line72 || $1 == line73) && first == 1)  \
   # The first message, print a separator, set first to two \
      { print "From oprnad@saupm00 Sat Jan 01 00:00 GMT 1995" \
        first = 2} \
   else { if (($1 == line72 || $1 == line73) && first != 1) \
   # Message is not the first: print two blank lines and a separator \
             { print "\n" \
               print "\n" \
               print "From oprnad@saupm00 Sat Jan 01 00:00 GMT 1995"} \
          else \
          # We're in a message \
             { if (substr($1,length($1),1) == COLON) \
             # Is the line a header? Print it and set CNTR to one \
                 { print $0 \
                   CNTR = 1 } \
               else \
               # We've finished the header \
                  { if (length($1) == 0) \
                  # Does a blank line follow the header? \
                  # If so, print it and set CNTR to two \
                     { print "\n" \
                       CNTR = 2 } \
                    else \
                    # We're in the body of the message \
                       { if (CNTR == 2) \
                       # Was the last line the blank line that \
                       # follows the header? \
                       # If so, print another blank line, the first \
                       # line of the body, and then set the CNTR \
                       # to one for the next line \
                          { print "\n" \
                            print $0 \
                            CNTR = 1 } \
                         else \
                         # We're really in the body, print it as is \
                            print $0  \
                       } \
                  } \
             } \
        } \
  }'  $inputfile > $inputfile.tmp

  # Clean the output file
  uniq $inputfile.tmp $outputfile
  # Remove the temporary file used for conversion
  rm -f $inputfile.tmp

  # Compress the old file to save space
  if ( -f ${notebookname}.notebook) then
    compress ${notebookname}.notebook
  endif
```

```
  echo Conversion finished
  echo $inputfile saved as $outputfile folder
endif  # of if !( -f $inputfile) then

echo " "
echo Press ENTER to return to the main menu ....
echo " "
echo " "
while ($< != "")
end
unset notebookname

breaksw

#
# Case 4 : NAMES to .addressbook conversion
#
case "4":
      set namesfilename
clear
echo " "
echo " "
echo -n 'Enter the name of the NAMES file: '
set namesfilename = $<
set inputfile = ${namesfilename}
set outputfile = ${HOME}/.addressbook
echo ''
echo 'NAMES file to be converted is: ' $inputfile
echo 'Output addressbook file is: ' $outputfile
echo ''

# Does the NAMES file exists in this directory?
if !( -f $inputfile) then
# No! Send the user a message
    echo 'File ' $inputfile ' does not exist'
else
# Yes :) OK start
   # Clearing tmp files if this is a rerun
   if ( -f tmp1) then
     rm tmp1
   endif
   if ( -f tmp2) then
     rm tmp2
   endif
   if ( -f tmp3) then
     rm tmp3
   endif
   # Check whether an addressbook already exists
   if ( -f $outputfile) then
   # there is one
```

```
     echo '----- Please NOTE -----------------------------------------
--'
     echo 'An addressbook already exists. '
     echo 'If you continue, it will be saved as .addressbook.bak'
     echo ''
     echo -n ' Do you wish to continue (y/n) : '
     set continue = $<
     if !($continue == "y") then
     # NO :(The user does not wish to continue.)
        echo " "
        echo Press ENTER to return to the main menu ....
        echo " "
        echo " "
        while ($< != "")
        end

        breaksw

     endif # of if !($continue == "y") then
   endif # of if ( -f $outputfile) then

   if ($continue == "y" || !( -f $outputfile)) then
   # Either the user wants to continue despite an existing
   # addressbook, or no addressbook currently exists
     if ($continue == "y") then
     # The user wants to rename their existing addressbook
       mv $outputfile $outputfile.bak
       echo $outputfile ' is saved as ' $outputfile.bak
     endif # of if ($continue == "y") then
     # START now
       echo ''
       echo Conversion started......
       # The following three awk statements are used for
       # reformating the NAMES file
       # tmp3 is used for conversion
       awk '{ \
             if( $0 ~ /.nick/) \
               {{print "\n" >> "tmp1"}  \
                { print $0 >> "tmp1"}} \
             else  \
               {print $0 >> "tmp1"} \
            }' ${inputfile}
       awk 'BEGIN {RS = ""; OFS= "\t"; FS="\n" } \
         { \
           {print"\n" >> "tmp2" } \
           {for (i = 1; i<= NF; i++) \
              printf("%s  ",$i) >> "tmp2" }  \
           {print"\n" >> "tmp2" }  \
         }' tmp1
       awk '{ \
             if( $0 ~ /.nick/) \
               { print $0,":" >> "tmp3"} \
             else  \
               {print $0 >> "tmp3"} \
            }' tmp2
       # Do the conversion
       awk '{ \
            enok=index($0,":nick.") \
            if ( enok > 0 ) \
               { nick=substr($1,7) \
                 uidd=substr($2,9) \
                 node=substr($3,7) \
                 namepos=index($0,":name.") \
                 if ( namepos > 0 ) \
                 # does the name tag has a value ? \
                    { \
                      namepos=namepos+6 \
                      entry=substr($0,namepos) \
                      enend=index(entry,":") \
                      enend=enend-1 \
                      name=substr($0,namepos,enend) \
                      print nick "\t" name "\t" uidd"@"node >>
"addressbook" \
                    } \
                 else  \
                 # Special case for VM/CMS system that used to be \
                 # SAUPM00 \
                     { if ( node == "" || node == "SAUPM00" || node ==
"saupm00") \
                         print nick "\t\t" uidd >> "addressbook" \
                       else \
                       # There is no value for the name tag \
                       print nick "\t\t" uidd"@"node >> "addressbook" \
                     } \
               } \
            }' tmp3
       # final result
       mv addressbook ${HOME}/.addressbook
       # clean temporary files
       rm tmp1 tmp2 tmp3
       echo ......Conversion finished
       echo ''
       echo $inputfile ' converted to ' $outputfile
   endif # of if ($continue == "y" || !( -f $outputfile)) then
endif  # of if !( -f $inputfile) then
if ($continue == "y") then
# There was already an addressbook and the user wanted to contiue
   echo ''
   echo '----- Please NOTE --------------------------------------------
'
```

```
   echo -n ' Do you want to merge the old addressbook with the new one
(y/n) : '
   set merge = $<
   if ($merge == "y") then
     cat ${HOME}/.addressbook.bak >> ${HOME}/.addressbook
     echo  ${HOME}/.addressbook.bak ' appeneded to ' ${HOME}/
.addressbook
   endif  # of if ($merge == "y") then
   echo ''
   echo '----- Please NOTE ---------------------------------------------
'
   echo -n ' Do you like to delete ' ${HOME}/.addressbook.bak ' (y/n) :
'
   set delete = $<
   if ($delete == "y") then
     rm -f ${HOME}/.addressbook.bak
     echo ${HOME}/.addressbook.bak ' been deleted'
     echo '------------------------------------------------------------
--'
   endif  # of if ($delete == "y") then
endif  # of if ($continue == "y") then
echo " "
echo Press ENTER to return to the main menu ....
echo " "
echo " "
while ($< != "")
end

breaksw

#
# Case 5 : Invoke Pine
#
case "5":
clear
pine

breaksw

#
# Case 6 : Exit the program
#
case "6":
set stop = 1
clear

breaksw

#
# In case someone entered any case other than one from above
```

```
#
default:
clear
echo " " ; echo " " ; echo " "
echo You have entered an INVALID CHOICE ....
echo " " ; echo " "; echo " "
echo Press ENTER to return to the main menu ....
while ($< != "")
end

breaksw

endsw
end
```

Iyad Al-Bukhari
Systems Administrator
King Fahd University (Saudi Arabia)

AIX 4.3.1's AutoFS feature

AutoFS is the newest addition to AIX's feature set. It's another component of SunSoft's ONC+ system management suite that's been adopted by AIX. CacheFS, another feature of ONC+ that was introduced in AIX 4.3, was the subject of an article of mine that was published in the April 1998 issue of *AIX Update*.

AutoFS's main function is to provide the automatic mounting of NFS file systems on machines that function as NFS clients. By automatically mounting file systems when access to them is needed and breaking the connections when they are not used for a pre-determined period of time, AutoFS reduces the network traffic needed to maintain an NFS connection.

AUTOFS COMPONENTS

AutoFS consists of three components:

- The **automount** command

- The AutoFS file system
- The **automountd** daemon.

The **automount** command is invoked at system start up from the file */etc/rc.nfs*. It reads the *auto_master* map file in order to build an initial set of AutoFS mounts. These mounts are not implemented automatically – they act as triggers, mapping file systems to be mounted from remote computers to local directories. As there are no AutoFS mounts in the mount table at boot time, all the AutoFS mounts defined in the master map are installed. The **automount** command may be run at any time after the booting to refresh the list of AutoFS triggers and bring it up to date. Deleted entries are unmounted and newly added entries are installed.

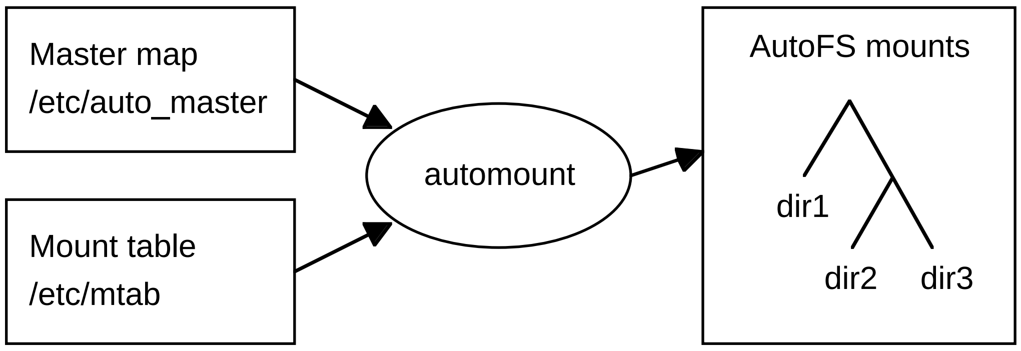

*Figure 1: **automount** and AutoFS*

When programs running on an NFS client try to access an unmounted file system, AutoFS intercepts the request, suspends the program, and calls **automountd** to mount the requested directory. The mount request contains all the information required to perform a mount: the map to be used, the key to be looked up, the path for the mount point, and the default mount options. The **automountd** daemon locates the directory in one of AutoFS's map files, performs the mount, and sends a reply to AutoFS. Upon reception of the reply, AutoFS allows the suspended program to resume. If the filesystem requested by the client is already mounted, the request is redirected by AutoFS without any involvement of **automountd**. AutoFS monitors the use of mounted file systems and sends unmount requests to the **automountd** daemon to unmount unused file systems. The **automountd** daemon is not required to keep any state information about AutoFS mounts. This means that the daemon can be killed and restarted without any effect on existing mounts.

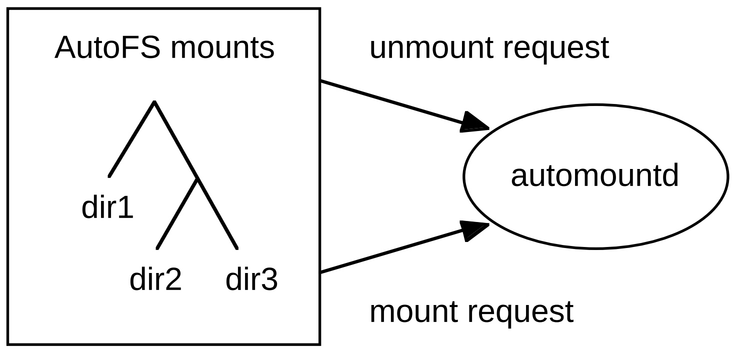

*Figure 2: AutoFS and **automountd***

AUTOFS COMMANDS

The two user-level commands that are used by AutoFS are **automount** and **automountd**.

automount

The **automount** command install AutoFS mount points and associates an automount map with each mount point. The **automount** command syntax is:

```
    automount [-t duration] [-v].
```

-t specifies the duration, in seconds, during which a filesystem is to remain mounted if not in use. The default duration is five minutes.

-v directs the command to produce verbose output information about mount and unmount commands issued by AutoFS and is used for troubleshooting.

automountd

The **automountd** daemon is an RPC server that processes and answers file system mount and unmount requests from the AutoFS file system. The **automountd** command syntax is:

```
automountd [-T] [-v] [-D name=value]
```

-T turns on tracing of RPC server calls, which results in the output of a trace to the standard output device.

-v displays verbose status and error messages on standard output device.

-D *name=value* assigns a value to the **automountd** daemon environment variable specified.

AUTOFS MAPS

AutoFS is configured by files known as ‘maps’. Maps can be implemented as local files or distributed by Network Information Service (NIS, formerly known as Yellow Pages or ‘YP’). AutoFS uses three types of map:

- Master maps
- Direct maps
- Indirect maps.

Master maps

The map */etc/auto_master* is a master list that specifies all the maps that AutoFS should know about. Each line of */etc/auto_master* conforms to the following syntax:

```
mount-point  map-name  [mount-options]
```

The above fields have the following meaning:

mount-point is the full (absolute) path name of a directory. If the directory does not exist, AutoFS creates it (if possible). If the directory exists and is not empty, mounting on it hides its contents. In such cases, AutoFS issues a warning message.

map-name is the map AutoFS uses to obtain directions to locations or mount information. If the name is preceded by a slash (/), AutoFS interprets the name as a local file. Otherwise, the file is assumed to be supplied by the NIS.

mount-options is an optional, comma-separated list of options that apply to the mounting of entries specified in ***map-name***. Any entries listed in ***map-name*** itself override these options. The options are the same as those for the standard NFS **mount** command, with the exception of the **bg** (background) and **fg** (foreground) options.

Below is an example master map, */etc/auto_master*.

```
#
# Master map for automounter
#
+auto_master
/-           /etc/auto_direct
/home        /etc/auto_home
```

After the comment lines, the first line in this file includes the map *auto_master*. If NIS auto_master file exists then it is included at this point. As the included master map is first in this master file, any entries in the included map override conflicting entries that follow in the local map. For instance, if the name server master map contains an entry for */home* that names a map other than */etc/auto_home*, then it takes precedence. A client can override entries in the name service map by placing entries before the included map entry.

Lines that begin with '/-' define direct maps that are explained in the next section. The line that begins with '*/home*' points to an indirect map.

Direct maps

A direct map defines a direct connection between a mount point on a client and a directory on a server. Lines defining a direct map have the following syntax:

```
mount-point  [mount-options]  location
```

mount-point is the path name of the local directory that serves as a mount location.

mount-options are mount command options for the particular mount.

location is the location of the remote file system specified as *server:pathname.*

Below is an example direct map, */etc/auto_direct*.

```
#
# /etc/auto_direct
#
/usr/man        -ro        host1:/usr/man
/usr/src        -ro        host2:/usr/src
/usr/local      -fstype=cachefs,cachedir=/cache,backfstype=nfs
                 host3:/usr/local
```

Note that the first two lines are mounted as read-only file systems and that the third one uses CacheFS.

Indirect maps

An indirect map performs a substitution based on the value of a key to establish an association between a local mount point and a remote directory on a server. Indirect maps are especially useful for accessing home directories. Lines that define indirect maps have the following syntax:

```
key  [mount-options]  location
```

key is a simple name (no slashes permitted) used in an indirect map.

mount-options are mount command options for the particular mount.

location is the location of the remote file system specified as *server:pathname.*

Below is an example of an indirect map, */etc/auto_home*.

```
#
# /etc/auto_home
#
mark                            host1:/home/mark
victor                          host1:/home/victor
alex        -rw,nosuid          host2:/home/alex
```

The use of the map */etc/auto_home* enables each user to mount his or her home directory automatically when they login to any machine that has NIS and AutoFS properly installed.

COMPATIBILITY ISSUES AND PROBLEMS

Although AutoFS is the default method of automatically mounting NFS under AIX 4.3.1, the previous version of **automount** is still supported. The command **/usr/sbin/automount** is a script that checks the environment variable *COMPAT_AUTOMOUNT*. If this variable is defined, the system uses the old version of **automount**, which runs as the command **/usr/sbin/automount**.

Note that the environment variable should be set before the invocation of the **automount** command. The main reason for simultaneous support for new and old versions of this command is to allow continuous operation of existing complicated **automount** set-ups. Another reason is the abundance of defects that still have to be ironed out of the new version. The following is a summary of known defects in AutoFS and their fixes (where available).

APAR Number	Defect Number	Description	Installation recommendation
IX76627	248091	Loader crashes machine when executing programs from AutoFS-mounted directories	Required
IX77023	248344	Memory leak caused by AutoFS when executing many mount/unmount requests	Recommended (stop and restart automountd as a temporary fix)
No APAR assigned	243313	Extra slashes appear in mount table output	Cosmetic
No APAR assigned	248200	When an NFS filesystem is automounted the df and mount commands show the hostname twice	Cosmetic

Fixes for this (and possibly other) defects should be available from IBM by the time this article appears in print.

REFERENCES

1 *The Automounter* by Brent Callaghan and Tom Lyon, Usenix Proceedings, Winter 1989.

2 *The Automounter: Solaris 2.0 and Beyond* by Brent Callaghan, SunSoft, October 1992.

3 *AIX Version 4.3 System Management Guide: Communications and Networks*, IBM Corp.

A Polak
APS (Israel)

CC-NUMA

This month's instalment is the second and concluding part of this article on CC-NUMA.

ADVANTAGES OF CC-NUMA

The CC-NUMA architecture offers a number of advantages for building high-order SMP systems. The following subsections discuss some of them briefly.

Maximize investment

Firstly, the basic building block of a CC-NUMA system, the module is generally based on an existing SMP system. This means that vendors can capitalize on their existing investment, which, in turn, results in a lower 'shop cost' and (hopefully!) a lower purchase price for the customer. This may also mean that customers can initially buy a single module that's later retro-fitted with NUMA to increase system size.

Balanced systems and 'pay-as-you-grow'

As each module comes with its own CPUs, memory, and I/O, adding a new module to an existing system brings with it each of these resources. This means that the system remains 'balanced' in terms of CPU capacity, I/O bandwidth, and memory as it grows. In other, non-

NUMA, systems, the customer must generally pay up-front for I/O bandwidth and for the infrastructure to be able to add more memory and CPUs later to expand the system to its maximum configuration, even though they may initially only have one or two CPUs installed. This means that the customer may buy performance features that are never used. With CC-NUMA, it's 'pay-as-you-grow'.

Improved availability

The inherent modularity of the NUMA architecture can be used in conjunction with reliability, availability, and serviceability (RAS) techniques to improve the 'uptime' of large systems significantly. For example, should memory, CPU, or another component on a module fail, then it is generally possible to re-boot the system without the module containing the defective part. In this way, the system continues to offer the service – albeit with a slight performance degradation – while the defective module is repaired. For this scenario to work, it's necessary for all the resources attached to the defective module (disks, network interfaces, etc) to have a second access path via another module.

Future proofing

Some systems, such as IBM's RS/6000 and Bull's Escala, are designed to allow CPUs to be upgraded as technology advances, so that, for instance, a PowerPC 601 may be upgraded to a PowerPC 604/604e or to a 64-bit PowerPC 620. But this ability is not widespread in the industry, and in general a move to the latest technology implies replacing existing hardware.

However, with an appropriately designed CC-NUMA system, it's possible to support different technologies (eg clock frequencies) on different modules in the same system. In this way, as the system grows, new modules may use the latest technology for CPUs and chip sets. For example, an initial configuration may be built using modules with 200 MHz processors with a 66 MHz system bus and EDO memory. A year or so later, the system is to be expanded and now the available technology is 400 MHz CPUs with 100 MHz system bus and SDRAM memory. The original technology may, indeed, be discontinued and no longer available. It is possible to have this mix of modules within a NUMA system as the module interconnect is asynchronous, and the system clock on each motherboard runs independently.

There are some subtle difficulties associated with this mixing of frequencies, particularly regarding the system clock and time. The operating system, and possibly the hardware, must be carefully designed in order to support this. For example, if a performance application is measuring throughput, it may read the time, T1, count a number of bytes transferred, say 1 MB, and read the time again, T2. It then uses the difference in time to calculate throughput. If this application is running on a 200 MHz CPU when it reads T1, and then, while counting, it is rescheduled on a 400 MHz CPU, if the calculation is to be meaningful at time T2, then the values must be coherent. Additionally, time must only go forwards – its value should not decrease when scheduled from a 400 MHz CPU to a 200 MHz CPU. One way round this is to use a common clock for the whole system, though this is inherently difficult, partly because accessing a clock on the same module carries a lower latency than accessing one on a remote module, and this variation in latency is generally higher than the precision required by the operating system. In fact this time problem is not limited to modules with different CPU clock speeds, but to all NUMA systems that use an asynchronous interconnect, since nominally identical clock frequencies are never identical, and therefore the time on individual modules drifts apart very quickly. This topic, though elementary, should be discussed with the system vendor.

Circumvent chip-set limitations

This topic is mostly limited to the ‘Wintel’ platform, but may be of interest to AIX users in a ‘mixed environment’ considering the future of their various platforms.

Intel’s SMP chip sets use a bus-based CPU interconnect, rather than the higher performance, but more expensive, crossbar-based solution adopted by most of the high-end system vendors. Because of this, Intel’s SMP systems are limited to two- or four-way systems. This is not a problem for Intel, which focuses on high-volume markets for desktops and low-end servers. Also, given that Windows NT, the primary OS for Intel SMP boards, does not scale beyond four CPUs (at least, not the shrink-wrapped product), increasing the number of CPUs from three to four produces little or no performance increase.

While it is difficult to combat the combined marketing machines of these two organizations, with today's technology and advanced chip sets, four-way modules are not the 'sweet-spot' in terms of either straight performance or price/performance. Many SMP vendors are now offering 10- or 16-way standard SMP systems, which maintain good scalability characteristics without resorting to NUMA. For optimum performance, combined with application and operating system simplicity, it is best to go as far as possible with the UMA (standard SMP) architecture before moving to NUMA.

However, as each module in a NUMA system comprises its own motherboard and chip set, NUMA enables system vendors to get round the Intel chip set limitation and build higher-order SMP systems.

Maintain the SMP architecture

As explained above, CC-NUMA is just a means of implementing SMP. Accordingly, all applications that run on existing SMP systems will run unchanged on the NUMA hardware. So a system vendor's existing application catalogue is also available for CC-NUMA-based systems.

Particularly suitable for certain types of application

The modularity of NUMA hardware, combined with the SMP paradigm, results in a system that's particularly suited to a certain type of application. These are the ones in which data and operations on data may be cleanly partitioned. A good example of this is data warehousing, where a large database is scanned from beginning to end to look for patterns. On a CC-NUMA system, each module can be given part of the database to examine, processing the subset in parallel with, and without interfering with, the other modules. Some of the best decision support benchmark figures, such as TPC-D, have been published using CC-NUMA systems. Other applications types, such as transaction processing, require more careful tuning, and probably won't yield such spectacular results.

WHAT TO LOOK FOR WHEN CONSIDERING NUMA

This section lists a few of the points to consider when evaluating a CC-NUMA system. The usual criteria, such as price/performance, industry standard benchmarks, applications catalogue, OS features (security, IPv6, Java, clustering), availability (uptime), service, support, cost of ownership, etc, still apply.

Full 64-bit support

CC-NUMA offers a means of assembling very high order SMP systems. In order for such systems to remain balanced and not become ‘memory bound’, each time a CPU is added it is necessary also to add memory. 32-bit systems are limited to 4 GB of physical memory and address space. With a 32-CPU system, 4 GB means just 0.125 GB per CPU. This is insufficient for today’s powerful CPUs, and the result is a system that spends a considerable amount of time swapping data to and from disk. Full 64-bit support means that the amount of real memory is limited only by the number of memory sockets available in each module and the depth of the customer’s wallet. Full 64-bit support means that it is possible to continue to increase the amount of real (physical) memory in proportion to the number of CPUs, and thereby maintain a balanced system.

While this is not strictly relevant to AIX, users in heterogeneous environments may like to bear in mind that some CPUs offer 36-bit addressing, providing for 64 GB of real memory, which – for today – is probably sufficient. However there remain a number of difficulties. Firstly the cache subsystems of these CPUs only cache across 32-bit addresses (ie 4 GB). This means that the remaining 60 GB cannot be cached in the L1 and L2 caches, which results in a performance degradation. Further, the operating systems running on these processors are 32-bit, and, as such, application address space is limited to 4 GB. As the real benefit of 64-bit computing is in the use of very large (greater than 4 GB) database buffers in physical memory, and given that the database application is still 32-bit, there is no performance increase. Some system vendors have tried to get round this problem through some complex modifications to the memory management routines, but these are both architecturally messy and introduce still more operating system overhead.

Investment protection

As discussed above, CC-NUMA can offer very good investment protection, though not all of today's systems provide for this. It is important to understand:

- Whether it is possible to replace CPUs within a module with the latest processors, for instance to migrate from Pentium Pro to Pentium-II to Merced, or from PowerPC 604 to PowerPC 620 or PowerPC RS64.
- Whether the system supports modules with different CPU, bus/crossbar, and memory frequencies, or modules with different numbers of CPUs. If this is not possible, then further system growth is not possible once the CPU becomes unavailable, making it necessary to replace the entire system.
- Whether 64-bit support is provided with the existing operating system or an OS upgrade is required to support 64-bit. If 64-bit support requires an OS upgrade, this may involve upgrading existing applications, which has sometimes been a delicate and expensive operation. You should also consider that, while AIX offers 64-bit support, some others, like Windows NT, don't.

NUMA FACTOR AND PERFORMANCE

The NUMA factor is an important figure to consider, as it gives an indication of both the overall system scalability and the amount of effort required to tune the OS and applications for the system.

In fact the NUMA factor isn't just one figure, as latency can be introduced in a number of places during a memory access. You should understand exactly what the numbers you are being given represent – are they best-case, average-case, or worst-case figures?

A NUMA factor of five or less will probably require no application modification, and will give reasonable scalability (NUMA scalability is not as good as that of UMA). A factor between five and ten is OK, but may require some application tuning. The implications for application software of a NUMA factor greater than ten should be discussed with the system vendor.

You should also understand the modifications and optimizations that have been made to the operating system to support NUMA, as described earlier.

Some vendors of NUMA hardware advertise support for up to 63 or 64 modules and up to 252 or 256 CPUs. Yet whenever they publish benchmarks, the systems used generally comprise eight or fewer modules. There are a number of reasons for this. Firstly, hardware scalability (increase in performance with each new module) diminishes as the number of modules increases, and (in some cases) system performance can actually decrease with the addition of extra CPUs. Secondly, the cost of a fully configured system is very high, hence the price/performance ratio suffers. And, thirdly, today's commercial and general-purpose software (operating systems and applications) usually doesn't scale beyond 16 to 32 CPUs (that's for operating system like AIX – Windows NT doesn't scale beyond four CPUs).

In order to achieve good application-level performance a Level-3 (L3) cache at the module level should be part of the system architecture. This L3 cache should be sufficiently large, given its degree of associativity, to ensure that it is effective and that the number of cache-line collisions is kept to a reasonable level given the size of the OS and application working sets.

As mentioned briefly earlier, for the same configuration (CPU-memory-I/O), standard UMA SMP will offer higher performance than the equivalent NUMA system. Because of this, it's best to take UMA as far as possible before turning to NUMA (remember you construct NUMA using individual UMA modules).

RELIABILITY

Reliability and availability are key issues, as more and more customers turn to continuous operation (24 hours a day, 365 days a year). As systems become ever more complex, the number of components that can go wrong increases and availability decreases. NUMA systems do not escape this rule. They are relatively complex both in terms of hardware and software, which means that, without particular attention to availability, downtime can become unacceptably high.

As with all high availability systems, the first requirement is to be able to insert the NUMA machine into a cluster. Since the standard SMP model is maintained, if the system provider already has a cluster solution, then it should run without difficulty in a NUMA configuration.

The operating system should include a number of availability features, such as being able to boot with only a subset of the available modules, to enable operation to continue while a defective module is repaired.

The interconnect should not be a 'single point of failure' – if there is only a single interconnect between modules, then the whole system stops if it fails. Having two interconnects avoids this problem, at the same time doubling the available inter-module bandwidth.

CONCLUSION

We have seen that CC-NUMA is a cost-effective way of building high-order SMP systems. The architecture has a number of advantages, but also generally requires some architecture-specific optimizations within the operating system and (possibly) applications. Because of the inherent characteristics of CC-NUMA systems, particular attention should be paid to a number of points when evaluating or comparing one CC-NUMA system with another, or with standard UMA SMP systems.

ADDITIONAL INFORMATION

- *http://www-frec.bull.com/OSBU2_0/biblio.htm* – bibliography of CC-NUMA literature.
- *http://www.lfbs.rwth-aachen.de/~marcus/SCI/links.html* NUMA links on the Internet
- *http://powderkeg.stanford.edu/OS/papers/SIGMETRICS95/abstract.html* – memory system performance of Unix on CC-NUMA multiprocessors.
- *http://www.almaden.ibm.com/journal/rd/413/kaeli.html* – performance analysis of a CC-NUMA prototype.

Jean-Paul Weber (France) © Xephon 1998

Using DHCP

INTRODUCTION

Networks are the nerves of today’s computer systems. They provide all the services today’s users rely on, such as e-mail, Web browsing, and other popular services on the Internet and intranets. These are not the only services to rely on networks – databases and other mission-critical applications are heavily network-dependent.

When it comes to AIX (and Unix in general), the principal network protocol is TCP/IP. Other operating systems have also adopted TCP/IP, often using it in preference to their own proprietary protocols. As TCP/IP is an open protocol, there is virtually no operating system nowadays that doesn’t offer TCP/IP support.

TCP/IP is based on IP addresses, which have to be unique in a network. But, as IP addresses are configured by humans, there is a chance of wrong configuration. This is not a big problem in small networks, by which I mean those with ten machines or so to manage. But for networks bigger than this, address assignment can be a pain. Furthermore, address assignment is not the only part of IP configuration that’s prone to error. There are a number of other attributes to configure for any given adapter, such as the subnet mask, default gateway, print server, time server, DNS server, and domain name, to name a few. All of these are also prone to error, and what can be done wrong will be done wrong given enough time or enough opportunities to do so.

Another important aspect to consider is that all these settings need to be made manually at each machine that needs to be configured. This places a considerable workload on network administrators at large shops, who have to go from machine to machine to configure or correct wrongly configured network adapters.

To address these needs, the Dynamic Host Configuration Protocol (DHCP) was developed through various RFCs (Request For Comments) as a successor to the old-fashioned Boot Protocol

(BOOTP), which was used for a long time to boot diskless workstations or X-Terminals. This protocol was, however, able to configure a network adapter only with a fixed IP address and a subnet mask and not much more.

DHCP offers a wide variety of options for configuring clients automatically with TCP/IP parameters, as long as clients support the protocol. And, in addition, AIX's implementation of DHCP also supports dynamic updates to the DNS server. This means that the network administrator now has one central point from which to configure IP parameters, and no longer has to configure the DNS server on its own.

GENERAL PROTOCOL FLOW

DHCP consists of a DHCP server daemon on the server machine, a DCHP client daemon on the client machine, and (optionally) a DHCP relay agent to transmit packets to a DHCP server if the server is not on the local network. Normally the DHCP agent would be the default router on the network, and, for backward compatibility, the configuration of a DHCP relay agent is the same as that of a BOOTP relay agent.

When a DHCP client is at the phase in its boot sequence at which it would normally configure its network interface, it sends a DHCP discover message. This is transmitted to the local network with an address of *255.255.255.255*, which is the 'local broadcast' address. In the packet it requests DHCP configuration. The packet is received by a DHCP server on the local network and is processed. If there is an address available for the client, the server constructs a DHCP offer message, containing the IP address and other appropriate options, and transfers it to the client. The client receives the offer and stores it while waiting for other offers. After choosing the best offer from those it received, the client broadcasts a DHCP request, specifying which server's offer it wants. All DHCP servers receive the request, but only the specified server returns a DHCP acknowledgement, assigning the reserved IP address and the specified options to the client. All other DHCP servers free their assignments for that client.

The acknowledgement signals that the transaction is complete and the client ‘owns’ the assigned IP address for a period of time specified in the server’s configuration file. This period is known as the ‘lease time’.

After half of the lease time has expired, the client sends a renew message to the server to extend the lease. If it doesn’t get an answer, it tries to find the server owning its IP address by broadcasting a DHCP rebind packet. This can happen if the server has been moved to another subnet. If the client can’t find the DHCP server, it uses the remainder of the lease time. Once this expires, the interface configured by DHCP is brought down and the client tries to find a new server by sending a new DHCP discover message. This procedure ensures the correct assignment of an IP address to only one client at any given time.

DHCP CONFIGURATION FILES

With AIX, there are three configuration files associated with the DHCP service:

- */etc/dhcpsd.cnf,* the server’s configuration file.
- */etc/dhcprd.cnf,* the relay agent’s configuration file.
- */etc/dhcpcd.ini*, the client configuration file.

If the package *bos.net.tcpip* was correctly installed on your system, then you should already have example configuration files installed. If not, then install them from CD-ROM by running **smitty install_latest**, providing the installation device/directory, looking for the string ‘DHCP’ in the listing, marking it, and installing the files found.

If you wish to use an AIX DHCP server to assign TCP/IP settings on Windows 95 or Windows NT clients, then configure the client via the Network Control Panel (choose the *Protocols* tab, then highlight the TCP/IP protocol, click *Properties*, and check the *Obtain an IP address from a DHCP server* radio button, if you’re running Windows NT, or the *Obtain an IP address automatically* radio button, if you’re running Windows 95).

Other operating systems should have equivalent means for configuring as DHCP clients.

CONFIGURING THE DHCP SERVER

The example DHCP server configuration file, */etc/dhcpsd.cnf*, is a bit long-winded, as it explains all the possible configuration options offered. Here is a breakdown of the important options.

Logging

First of all, it's important to monitor what the DHCP server is doing. For this reason logging should be the first thing to be configured. Any item to be logged is put into the log file specified by a line similar to the one below.

```
    logFileName      /var/tmp/dhcpsd.log
```

The number of rotating log files can be specified with:

```
    numLogFiles      4
```

The length of each log file (in KB) can be set using:

```
    logFileSize      100
```

It is also possible for you to specify the various items to be logged – for example:

```
    logItem          INFO
    logItem          EVENT
    logItem          ACTION
    logItem          ACNTING
    logItem          WARNING
    logItem          SYSERR
    logItem          OBJERR
    logItem          PROTERR
    logItem          TRACE
```

I think the format of these entries is fairly self-explanatory. Any event that occurs with the DHCP server is then shown in the log along with an appropriate label.

Lease duration and BOOTP

For the parameters that follow, you should bear in mind that the scope of any parameter is defined by its placement. Thus, for instance, if a parameter is defined outside all curly braces, it is valid for all networks, subnets, and classes of clients, as well as for all clients. It's usually a good idea to define some items outside all network definitions so as to set standard values, though it's also possible to define different values for specific subnets or classes of client. For instance, mobile computers benefit from having different settings from, say, desktop workstations. This means that a normally long-leased address can be assigned to them for a shorter period. If not defined within the scope of a network, subnet, class, or client, a value becomes a default.

The value to define the lease period is:

```
leaseTimeDefault        2 weeks
```

The default unit for this is minutes (if no unit is specified), and you may also define the value in terms of years, months, weeks, days, hours, and seconds. The value has to be specified as a decimal number.

Next, the check interval for the lease time has to be specified. After this period has expired, the server looks for expired leases and frees them if the client has not renewed them. The address is then available if it's needed by another client.

It's important to note that this parameter is valid for any lease given out by this server and should therefore be based on the default lease period. In addition – and just as importantly – note that this parameter applies to *all* leases given out by this server. This means that it cannot be overridden by any client!

```
leaseExpireInterval     5 days
```

As DHCP is the successor of BOOTP, it also serves BOOTP clients. To activate this feature, set the parameter below.

```
supportBOOTP            yes
```

Any word other than 'yes' causes the server not to serve BOOTP clients.

You can also configure your DHCP server to support clients not explicitly listed in your configuration files. Once this is done, any client sending DHCP messages will be served with an IP address by the server. As this may not be what you want, consider the implications of this statement before deciding whether you want to support unlisted clients. If not specified, *any* client will get an IP address.

```
supportUnlistedClients  no
```

OPTIONS

Now the preliminaries are over, it's time for configuration examples. For any given network, subnet, or class, you can specify 'options' to be set from a range of some 60 available ones. Have a look at the sample *dhcpsd.cnf* file for a full list of options. All options are identified by a unique number, the first one being 0 (zero), the Pad option. Some basic options need not be specified, such as Request IP address (option 50), Overload (option 52), and DHCP Message Type (option 53), as they are implicitly understood. Other options need to be explicitly set, and some of the most important ones are discussed below. As always, your interpretation of what's important may be different from mine. Again, have a look at all options specified in *dhcpsd.cnf* or in the **man** page for **dhcpsd**. Options that I use are (I've left them in the normal notation in which they'd appear in the configuration file):

```
     option 1          255.255.255.0
```

Option 1 is the subnet mask to be used by the client.

```
     option 3          192.168.100.1
```

Option 3 specifies the host that's the default router (or 'gateway' in TCP/IP terminology) for this subnet.

```
     option 6          192.168.100.3
```

Option 6 specifies the DNS server to be used.

```
     option 9          192.168.100.3
```

Option 9 specifies the print server (LPR), which in this instance is also the DNS server.

```
     option 15         "mydomain.com"
```

Option 15 specifies the domain name. Replace the value in quotes with the one appropriate to your network.

Some of the options can be assigned per host. Here are some examples:

```
     option 19         0
```

Option 19 specifies that IP forwarding is true – this host is a router.

```
    option 23         128
```

Option 23 specifies the default IP packet TTL (Time To Live).

Other options can be specified per interface:

```
    option 28         255.255.255.255
```

Option 28 specifies the local broadcast address.

```
    option 33         194.200.124.0 192.168.100.7
```

Option 33 is used to specify static routes. Packets destined for network 194.200.124 won't go through 192.168.100.1 (the default router specified above) but through 192.168.100.7 instead.

```
    option 35         300
```

Option 35 controls when ARP cache entries time out. The unit is seconds. Other useful parameters for applications would be:

```
    option 40         "mynisdomain.dom"
```

Option 40 specifies the NIS domain to use.

```
    option 42         "timeserver.domain.edu"
```

Option 42 specifies the NTP time server to use for network time.

```
    option 44         192.168.100.127
```

Even for AIX users this is sometimes unavoidable: option 44 specifies the WINS server to use for NetBIOS name resolution.

```
    option 46         2
```

Option 46 specifies the NetBIOS over TCP node type. Type 2 is the so-called 'hybrid node' or 'h-node'.

```
    option 48         192.168.100.3
```

Option 48 specifies the 'font server' – our DNS and LPR server also serves X11 fonts to clients.

```
    option 49         192.168.100.8
```

Option 49 specifies the XDMCP server. If configured correctly, it opens a chooser on clients that offers graphical login via X11.

To support BOOTP, some specific options can be configured:

```
    option sa         192.168.100.100
```

This is the server address for the BOOTP client to use.

```
    option bf         "/tftp/bootfile.client1"
```

The boot file for the specified client to use.

```
    option hd         "/home/bootclient1"
```

The home directory for the client using BOOTP

As you can see, there are many options that can be specified. But don't be put off by this – you normally need to set only a handful of the available ones to configure clients. More important is the configuration of the networks and subnets, as shown below.

NETWORKS, SUBNETS, AND CLASSES, AND THEIR SCOPE

You define networks that DHCP is to serve by network address and (optionally) subnet mask and range of addresses. The network address is the IP address of the network, specified in dotted quad notation. When specifying the address, you should respect its network class, so that, for instance, a class A address should not specify any subnets: instead of 10.25.175.0, use 10.0.0.0, specify a subnet mask, and define the subnet(s) to serve within the scope of the network. If using one network, you could specify:

```
    network 10.0.0.0
    {
    }
```

This includes all 16 million or so addresses in this network in the scope, indicating that the DHCP server is to serve them all. No options are specified. Clients receive IP addresses starting with 10.0.0.1, and also receive all options specified outside the network scope. A better way would be to specify the network with a class B subnet mask, which allows you to define 254 subnets each with some 65,000 hosts:

```
    network  10.0.0.0  255.255.0.0
    {
    }
```

This implies that you add at least one subnet scope to the definition like this:

```
network  10.0.0.0  255.255.0.0
{
     subnet  10.1.0.0  10.1.1.1-10.1.254.254
}
```

Now the server gives out addresses 10.1.1.1 to 10.1.254.254 to clients requesting an address. More subnets can be specified, each with its own range of host addresses. More to the point, each subnet can have its own set of options. If set up this way it is possible for every subnet to have its own router, which is specified by the router option:

```
subnet 10.1.0.0
{
     option 3     10.1.1.15
}
```

This would interfere with the range specified above (because of the default router's addresses). So what do you do? You could exclude its address from those being served by excluding it from the range:

```
network  10.0.0.0  255.255.0.0
{
     subnet  10.1.0.0  10.1.1.20-10.1.254.254
}
```

Now only the addresses above 10.1.1.20 are served and those below the specified range are free to be assigned manually. This is very handy if your network design includes routers and other servers, such as DNS servers, that have static IP addresses in every network. Even if it doesn't, options can be specified for all subnets separately, allowing you to configure appropriate servers and services for each subnet. Consider the following example:

```
network 10.0.0.0 255.255.0.0
{
     subnet  10.1.0.0  10.1.1.20-10.1.254.254
     {
     option 3     10.1.1.15          # default router
     option 6     10.1.1.10          # DNS server
     option 15    "mydomain.com"     # domain name
     }
     subnet  10.2.0.0  10.2.1.20-10.2.254.254
     {
     option 3     10.2.1.15          # default router
     option 6     10.1.1.10          # DNS server
     option 15    "mydomain.com"     # domain name
     }
}
```

In the above example, both subnets have the same DNS server and domain name, which is therefore a suitable candidate to put in the network scope:

```
network  10.0.0.0  255.255.0.0
{
option 6     10.1.1.10          # DNS server
option 15    "mydomain.com"     # domain name

     subnet  10.1.0.0  10.1.1.20-10.1.254.254
     {
     option 3     10.1.1.15          # default router
     }

     subnet  10.2.0.0  10.2.1.20-10.2.254.254
     {
     option 3     10.2.1.15          # default router
     }
}
```

Networks can quickly become extremely complicated. Sometimes you reserve some subnets for hosts of a specific class, such as Windows PCs. An example of how to serve addresses to this class of client is shown below.

```
network  10.0.0.0  255.255.0.0
{

option 6     10.1.1.10          # DNS server
option 15    "mydomain.com"     # domain name

     subnet  10.1.0.0  10.1.1.20-10.1.254.254
     {
     option 3     10.1.1.15          # default router
     }

     subnet  10.2.0.0  10.2.1.20-10.2.254.254
     {
     option 3     10.2.1.15          # default router
          class netbios_hosts
          {
          option 44     10.2.1.7     # NetBIOS over TCP server
                                     # (aka WINS)
          option 46     2            # NetBIOS over TCP node
                                     # type
          }
     }
}
```

CLIENT-SPECIFIC CONFIGURATION

The examples above concern the configuration of a whole network or subnet. Sometimes it is necessary to configure specific clients with a set of options instead. This could start with a specific IP address to be given to the client and end with configurable options. Client addresses can be given within any scope, be it a network, subnet, or class. We will start with some simple examples.

This entry states that IP address 10.2.1.135 should not be served:

```
client  0  0  10.2.1.135
```

As you see, the keyword 'client' is followed by a number, followed by a white space, followed by a second number (in this instance). The first number is the ID type. A zero (0) denotes a string, any other number is interpreted as type of hardware defined in RFC1340. If the second number is zero (0), this IP address is not served.

Sometimes it is desirable to serve IP addresses based on the hostname of the client:

```
client 0 "hostname"
```

or the client's MAC address:

```
client 6 40007A9C73B4
```

The first number is 1 for Ethernets and 6 for Token Rings, for example. These are used to distinguish between different hardware types of network adapter. It is followed by the MAC address in hexadecimal notation without a leading '0x'.

As with networks, subnets, and classes, a client can also have a scope. Within this scope all the options mentioned above can be specified, for example:

```
client 6 40007A9C73B4
{
     option 3 10.2.1.17
     option 6 192.168.100.3
     option 9 192.168.100.7
}
```

This says that the client with the IP address assigned by the associated network (10.2.0.0) is to use the DNS server at 192.168.100.3 and the

LPR server at 192.168.100.7, thus printing across a network boundary and using a DNS not in its own network. While this type of configuration is possible, in my experience it's extremely rare. Nonetheless, let's put together all the pieces to form a valid */etc/dhcpsd.cnf* file. This would look as follows:

```
sample /etc/dhcpsd.cnf:

logFileName     /var/tmp/dhcpsd.log
numLogFiles     4
logFileSize     100
logItem         INFO
LogItem         EVENT
logItem         ACTION
logItem         ACNTING
logItem         WARNING
logItem         SYSERR
logItem         OBJERR
logItem         PROTERR
logItem         TRACE

leaseTimeDefault           2 weeks
leaseExpireInterval        5 days
supportBOOTP               yes
supportUnlistedClients     no

network 10.0.0.0  255.255.0.0
{
     option 6     10.1.1.10          # DNS server
     option 15    "mydomain.com"     # domain name

     client 0  0  10.2.1.135
     client 0  "hostname"

     subnet 10.1.0.0  10.1.1.20-10.1.254.254
     {
          option 3     10.1.1.15          # default router
     }

     subnet 10.2.0.0  10.2.1.20-10.2.254.254
     {
          option 3     10.2.1.15          # default router

          client 6  40007A9C73B4
     {
          option 3     10.2.1.17
          option 6     192.168.100.3
          option 9     192.168.100.7
     }
```

```
            class netbios_hosts
            {
                 option 44    10.2.1.7      # NetBIOS over TCP
                                            # server (aka WINS)
                 option 46    2             # NetBIOS over TCP
                                            # node type
            }
       }
  }
```

And it says:"Logging is enabled in the file */var/tmp/dhcpsd.log,* and four log files are to be maintained in rotation, each with a maximum size of 100 KB. Any event, error, or notice recorded by the server, or action taken by the server, is to be logged. The default lease time is two weeks, but after five days the server scavenges unused IP addresses and releases them if they are free. The server is to support BOOTP clients, but will only serve them with an IP address; options are to be given to them based on the subnet in which they are located. The server will support unlisted clients and give them IP addresses from the range 10.1.1.20 to 10.1.254.254. The default DNS server is 10.1.1.10, but the client with MAC address 40:00:7A:9C:73:B4 will use the DNS server at 192.168.100.3. The domain name for all clients is '*mydomain.com*'. The IP address 10.2.1.135 will never be served. The client with hostname '*hostname*' will get a pre-specified IP address. Clients in subnet 10.1.0.0 have the default router 10.1.1.15, while clients in the subnet 10.2.0.0 use 10.2.1.15 as their default router. All clients in subnet 10.2.0.0 requesting option 44 and/or option 46 will be in the class *netbios_hosts* and will use the WINS server at address 10.2.1.7 and are to be set up as NetBIOS node type 2 (h-node), calling WINS first before trying to resolve NetBIOS names using DNS. The client with MAC address 40:00:7A:9C:73:B4 is to get special treatment – it is to use the DNS server at 192.168.100.3 for name resolution and print to the print server at network address 192.168.100.7. Its gateway is to be the host at 10.2.1.17."

This article has described the customization of the */etc/dhcpsd.cnf* DHCP server configuration file. In a future article I intend to describe how to customize the DHCP relay agent file and the DHCP client file.

Peter Wuetefeld
ResNova Unternehmensberatung GmbH (Germany)

AIX and IBM RS/6000 books

An excellent source of information on AIX and the RS/6000 are the many books written by AIX experts who often have no formal connection to IBM. Their connection to AIX is that they use it, and, as such, may truly call themselves AIX power users. They share their hard-gained information and experiences through the books they author. A description of four such books follows.

- *AIX Performance Tuning*
  Frank Waters
  Prentice Hall PTR
  ISBN 0-13-386707-2

  AIX contains many features to improve performance. *AIX Performance Tuning* describes ways in which these features can and should be used, and also discusses some of the trade-offs and potential pitfalls. Key topics include:

  – Basic concepts of performance measurement and tuning.

  – Why small changes can make big differences in performance.

  – Designing and implementing systems for performance.

  – How to analyse and tune existing systems and programs for maximum CPU, memory, disk, and communications performance.

  – How to interrogate, set, and tune all the major AIX performance parameters.

- *AIX RS/6000 System and Administration Guide*
  James DeRoest
  McGraw-Hill
  ISBN 0-07-036439-7

  AIX RS/6000 System and Administration Guide shows you:

  – How AIX compares to Berkeley and System V Unix and how to migrate AIX into an ‘open systems’ environment.

   – How to manage complex distributed systems, including workstation cluster topologies, network batch queueing, high-speed networks, and recovery procedures.

   This comprehensive book is also filled with detailed command descriptions, examples, diagrams, and troubleshooting tips, and includes a quick outline and keyword glossary – everything you need to make the most of AIX for RS/6000.

- *AIX/6000 Developer's Toolkit*
  Kevin Leininger
  McGraw-Hill
  ISBN 0-07-9119933-X

  This book is an indispensable guide for all application developers working in an AIX/6000 workstation environment. Included is a CD-ROM with all relevant source code, plus binaries for both AIX 3.2.5 and AIX 4.1.

- *The AIX Survival Guide*
  Andreas Siegert
  Addison-Wesley
  ISBN 0-201-59388-2

  The AIX Survival Guide is a comprehensive guide to the inner workings of AIX. Filled with much practical detail and many helpful hints and pointers to potential problems and pitfalls, this book is a valuable guide for AIX administrators and power users. For experienced Unix administrators, *The AIX Survival Guide* is a useful road map of the peculiarities of AIX systems.

I find all four of these books contain worthwhile information. Certainly most of their information is already common knowledge to me and I browse a large part of their contents. However, they all contain enough new information to make them worth reading. I use this newly gained information to help me with my customers who expect me to solve their daily business problems and emergencies, and help them develop plans for future computer systems to achieve their business goals.

AIX Specialist (Switzerland) © Xephon 1998

November 1995 – October 1998 index

Below is an index of all topics covered in *AIX Update* since Issue 1, November 1995. The numbers in **bold** are issue numbers, and the numbers in brackets are page numbers. If you'd like to order back-issues of *AIX Update*, please contact Xephon. You'll find contact information listed on page 2 – back-issues are available from Issue 1, and can be obtained from any of Xephon's offices.

Subject		**Issue** (page)
ADA		**15** (40)
addbib		**7** (33-35)
ADSM	*Managing*	**15** (12-18)
	Storage	**6** (43), **16** (22-31)
	Tape volumes	**2** (24)
AIX	*Enhanced facilities*	**6** (14-19)
AIX 4.1		**2** (3-4)
	Overview	**1** (32-33)
AIX 4.1	*Printing*	**3** (8-9)
AIX 4.1.4	*HACMP*	**15** (3)
	man	**13** (3-6)
AIX 4.1.5		**25** (16-26)
AIX 4.2		**22** (43-54)
	Overview	**11** (3-10)
AIX 4.3	*64-bit architecture*	**34** (33-45)
	Preview	**22** (8-11)
AIX BOS	*Performance tuning*	**21** (3)
AIX Connections II		**22** (53)
AIX hot fixes		**26** (46-47)
AIX PDT	*Performance tuning*	**21** (3)
aixterm		**33** (34-37)
Archival	*ADSM*	**16** (25)
arp		**7** (7-8)
AS/400	*Communication*	**9** (32-39)
at		**13** (43-45), **18** (19-23)
audit	*Script logging*	**17** (9)
AutoFS		**36** (17-24)
Automatic documentation		
	Utility	**9** (40-41)

Subject		**Issue** (page)
Availability		**7** (37)
awk		**8** (9-25)
Back-up		**6** (40-41)
	ADSM	**16** (24-25)
	AIX 4.1	**2** (3-4)
	Tape	**7** (22-32)
	Utility	**1** (21-29), **6** (10-14), **9** (9-27), **20** (3-18)
backup		**7** (22-26)
batch	*Sheduling jobs*	**13** (43-45)
Batch jobs	*Management*	**5** (40-41)
Best/1	*Performance tuning*	**21** (3)
bison		**15** (39)
Bonus Pack	*AIX 4.2*	**11** (4)
bsh		**2** (6), **4** (22), **20** (45)
Bull Advanced Server		**7** (40-41)
C/C++ compilers		**15** (39), **18** (41-43)
CA-Unicenter		**6** (32-33)
CacheFS		**30** (3-11)
CC-NUMA		**35** (30-39), **36** (24-31)
CDE	*AIX 4.2.1*	**22** (44-45)
chdev	*System parameters*	**1** (40)
_check_user		**33** (37-39)
chps		**22** (5-6)
CICS		**12** (11-48)
CICS for AIX		**2** (7-23), **3** (10-18)
	Resources	**6** (3-9)
cleantab		**15** (29-36)
Client/server		**29** (30-38)

Subject		**Issue** (page)
HACMP		**15** (3-11), **32** (22-40)
	Components	**15** (6-10)
HAGEO		**18** (27-32)
Hard and symbolic links		**20** (34-39)
HP-OpenView		**6** (34)
IBM 7133 disk array		**13** (26)
IBM Digital Library		**20** (27-34)
ifconfig		**7** (8-9)
iFOR/LS		**9** (3-9)
inetd	*Performance tuning*	**1** (18)
Informix-OnLine		
	ADSM	**15** (17)
inode		**20** (35)
Internet		**10** (6-10)
	AIX resources	**20** (18-23)
	E-mail	**18** (3-17)
iostat		**1** (11), **20** (43-44)
IP addresses		
	TCP/IP	**4** (28-31)
	Utility	**19** (44-47)
ipreport		**7** (14)
iptrace		**7** (14)
JDK	*AIX 4.2.1*	**22** (54)
Job monitoring		
	Utility	**21** (28-34)
Kapture/6000		
	AIX 4.2.1	**22** (53)
kill		**10** (41-43), **35** (39-43)
Korn shell		**25** (52-55)
ksh		**2** (6), **4** (22), **20** (45)
	Command history	**10** (3-6)
	Command retrieval	**11** (19-23)
	Cursor keys	**14** (8-9), **16** (42)
Licence management		
	iFOR/LS	**9** (3-9)
limit		**4** (4)
LISP		**15** (40)
locate	*Finding files*	**17** (43)
logger		**31** (17-18)
Login	**adm**	**8** (40)
	Customization	**7** (38-39)
lookbib		**7** (33-35)
lp	*Error messages*	**17** (5-8)
ls	*Advanced options*	**13** (32-43)
lsattr	*System parameters*	**1** (40)
lsps		**22** (6)
lssrc		**7** (13)
lsuser		**26** (49-51)
LVM		**6** (17-19)
	AIX 4.2	**11** (5-6)
man		**13** (3-6)
MAXM		**6** (34)
Memory		**4** (3-8), **4** (42), **8** (35-39)
	CICS for AIX	**2** (16-18)
Menus		**7** (42)
	Utility	**3** (3-8), **4** (8-22), **28** (3-12)
MIME		**18** (10-11)
mkpasswd		**22** (15), **30** (36-37), **30** (36-37)
mkps		**22** (5)
mksysb	*AIX 4.2*	**11** (6)
Model number		
	Utility	**5** (27-28)
Modems		**9** (27-32)
Monitoring AS/400s		
	Utility	**14** (3-7)
Monitoring using PCs		
	Utility	**23** (3-32), **24** (12-38), **27** (42-45), **28** (18-34), **29** (39-54)
MTA		**18** (3-10)
Netmanage	*FTP*	**15** (24-28)
netstat		**7** (10-13)
NetView for AIX		**6** (35-36)
	PTFs	**1** (37)
	SNMP	**11** (23-35)
NetWare		**2** (42-43)
Network	*Utility*	**31** (38-41)
Network Station		**23** (35)
NFS	*AutoFS*	**36** (17-24)
	CachFS	**30** (3-11)
nice		**1** (11), **22** (13-14)
NIM		**11** (7-8), **25** (30-41)
nslookup		**7** (13)
ODM (Object Data Manager)		**1** (39-40)
OpenGL	*AIX 4.2.1*	**22** (52)
Oracle	*ADSM*	**15** (14-16)

Subject		**Issue** (page)
oslevel		**4** (41), **25** (17)
p2c		**15** (40)
pack		**14** (50-51)
Paging space		
	Utility	**18** (23-27)
	Management	**22** (3-8)
Pascal		**15** (40)
pdt	*Performance*	**3** (33-42)
Performance		**1** (10-20), **21** (3-17), **32** (3-16)
	3dmon	**29** (7-8)
	Client/server	**28** (35-47)
	I/O pacing	**34** (3-7)
	monitor	**26** (39-45)
	Tuning	**27** (17-34)
	Utility	**10** (10-25), **19** (12-27), **25** (3-12), **35** (10-21)
	xmperf	**29** (3-14), **31** (21-38)
PerformanceWorks		**6** (34-35)
PerfPMR		**21** (5)
PEX	*AIX 4.2.1*	**22** (53)
PHIGS	*AIX 4.2.1*	**22** (53)
ping		**7** (5-7)
POP		**18** (12)
POWER		**3** (22-32)
PPP	**bos.net.ppp**	**22** (31)
	Implementation	**22** (25-43)
pppstat		**25** (24-26)
Printing	*AIX 4.1*	**3** (8-9)
	Form overlays	**26** (21-36)
	Header and trailer pages	**33** (40-51)
	Utility	**8** (25-27), **8** (40-42)
	Performance tuning	**1** (19)
Printing problems		
	Utility	**21** (17-27)
Processes		**4** (27)
	kill	**35** (39-43)
	Management	**10** (36-43)
	Utility	**12** (3-5), **31** (41-43)
ps		**1** (11), **10** (37-40), **22** (15)
PTX		**21** (3-5)

Subject		**Issue** (page)
PTX (continued)		
	AIX 4.2.1	**22** (53)
Remote execution		**35** (21-29)
renice		**22** (13-14)
REXX	*Utility*	**1** (30)
RMON		**3** (36-41)
rmss		**22** (16)
RPC	*CICS for AIX*	**2** (9)
RS/6000	*Model 390*	**4** (43)
	Model 70	**27** (47-51)
	Model F50	**23** (33), **31** (46-47)
runsrv	*Monitoring servers*	**15** (37-39)
sar		**1** (11)
Scalability		**1** (7)
schedtune		**21** (8-13), **22** (14-17)
Script logging		
	Utility	**17** (9-21)
SCSI	*Versus SSA*	**28** (12-17)
Security		**7** (36-37), **8** (28-31)
	Passwords	**24** (43-45)
	Root user	**34** (7-16)
	Utility	**26** (3-11)
sed		**18** (32-40)
sendmail		**18** (3-17)
setrlimit		**4** (5)
shutdown		**26** (48-49)
skulker		**15** (29)
SLIP		**22** (25-26)
smit		**6** (15-17), **16** (16-22)
	Management	**5** (41-43)
	Utility	**8** (27-28)
SMP	*CICS for AIX*	**3** (14-15)
	Performance tuning	**1** (15-16)
	POWER	**3** (26-27)
SMTP		**18** (3-17)
SNA		**3** (18-21), **5** (3-10)
SNA Server		**3** (18-21), **5** (3-4)
SNA Services		**3** (18-21), **5** (3-4)
SNMP		**3** (33-42)
	Using traps	**11** (23-35)
Socket programming		**13** (6-25)
Sockets		**12** (12-16), **16** (33-41), **17** (21-40), **19** (27-43)

Subject		**Issue** (page)
split		**31** (43-46)
splitlvcopy		**25** (22-25)
SQL-BackTrack		
	ADSM	**15** (14)
SSA		**11** (35-42), **13** (26-32), **13** (27-32)
	HACMP	**15** (7)
	Performance	**13** (31-32)
	Protocol	**13** (29-30)
	Reliability	**13** (31)
	Versus SCSI	**28** (12-17)
su		**18** (17-19), **25** (13-16)
suid		**6** (42)
	Security problems	**16** (31-32)
svmon		**22** (15-16)
Swap space		**1** (16)
swapon		**22** (5)
Sybase	*ADSM*	**15** (13-14)
syslogd	*Security*	**8** (28-31)
System configuration		
	Utility	**30** (11-25)
System management		**1** (7), **6** (31-36), **7** (36)
	iFOR/LS	**9** (3-9)
System monitoring		
	Utility	**19** (3-11)
System parameters		**1** (40-42)
SystemGuard		**32** (40-42)
SystemView for AIX		**6** (35-36)
Tape		**6** (36-39)
Tape management		
	Utility	**7** (15-32)
tar		**7** (22-26)
TCP	*Performance tuning*	**1** (18)
TCP/IP		**2** (41-42)
	IP addresses	**19** (44-45)
	Routing	**4** (28-37)
	Troubleshooting	**7** (3-15)
TCPAccess	*Programming*	**16** (33-41)
Thread		**35** (3-10)

Subject		**Issue** (page)
timejob	*Utility*	**21** (28-34)
timex	*Performance tuning*	**1** (10)
TME		**6** (36)
Token Ring	*CICS for AIX*	**3** (19-20)
tprof		**22** (12)
traceroute		**7** (9-10)
UDP	*Performance tuning*	**1** (18)
uname	*Model number*	**1** (42)
uncompress		**14** (50-51)
Unix to CICS transfer		
	Utility	**13** (6-25), **14** (9-35)
unlimit		**4** (4)
uptime	*Performance tuning*	**1** (11)
User admin	*Utility*	**27** (34-41)
uucp		**24** (38-42)
vi		**5** (17-27)
	Advanced features	**16** (3-16)
	Editing mulitple files	**6** (42-43)
	Search and replace	**23** (36-58)
Virtual Memory Manager		**21** (8-11)
Visualization Data Explorer		**6** (21-31)
vmstat		**1** (11), **20** (39-43)
vmstat		**22** (15)
vmtune		**21** (10-13), **22** (17-19)
Volume groups		**4** (23-25)
	Utility	**25** (27-29)
VSM		**6** (15-17)
WAN		**9** (27-32)
whoami		**25** (13)
X Server	*Communication*	**2** (43)
X-Terminals		
	xdm	**8** (31-35)
X-Windows		
	Performance tuning	**1** (18)
X/Motif		**26** (11-20)
xdm		**8** (31-35)
xlC		**18** (41-43)
yacc		**15** (39)
Year 2000		**24** (45-48)
zcat		**14** (50-51)

AIX news

IBM has announced Global Sign-On for Multiplatforms Version 2.0, a GUI-based single sign-on tool that coordinates user logons to distributed resources with a single authentication. It's designed to use TME 10 User Administration and TME 10 Software Distribution and runs on AIX 4.2.*x* servers (also supported are servers running Solaris 2.5.1 and NT 4.0) and on a range of Windows and OS/2 clients. Prices weren't announced.

The company also announced MQSeries Workflow 3.1, the follow-on version of its FlowMark business process automation and workflow management software. Out now on AIX and Windows 95 and NT, prices start at US$25,000, with upgrades from FlowMark 2.3 starting at US$12,500.

For further details, contact your local IBM representative.

* * *

Inrange Technologies, formerly General Signal Networks, has announced Link/9000 PCI-to-ESCON adapter card for AIX, for data transfer between RS/6000 systems, including SP systems,and S/390s. The company also announced ChannelDrive, a high-speed software driver offered as an option to the Link/9000, promising to speed up data transfer operations in AIX servers. ChannelDrive doesn't require a TCP/IP stack, which helps streamline the transfer process. Prices weren't announced.

For further details contact:
General Signal Networks, One Waterview Drive, Shelton, CT 06484, USA
Tel: +1 203 926 1801
Fax: +1 203 924 6400
Web: http://www.gsnetworks.com

General Signal Networks, Enterprise House, Foundation Park, Maidenhead, Berkshire SL6 3UD, UK
Tel: +44 1628 822244
Fax: +44 1628 822640

* * *

Chili!Soft, which makes enabling technology for cross-platform Active Server Pages (ASP), has signed a deal with IBM to deliver its Chili!ASP for AIX on RS/6000 servers. The software, we're told, will allow developers to build ASP Web applications on any operating system and deploy them on RS/6000s. The software is used for building and maintaining Web sites and applications. Chili!ASP for AIX will be available Q4, but no details on prices.

For further details contact:
Chili!Soft, Richards Road, Suite 103, Bellevue, WA 98005, USA
Tel: +1 425 957 1122
Fax: +1 425 562 9565
Web: http://chilisoft.com

xephon