28

AIX update

February 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 75067
USA
Telephone: 940 455 7050

Australian office

Xephon/RSM
GPO Box 6258
Halifax Street
Adelaide, SA 5000
Australia
Telephone: 08 223 1391

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.

Editor

Harold Lewis

Disclaimer

Subscriptions and back-issues

A year's subscription to *AIX Update*, comprising twelve monthly issues, costs £170.00 in the UK; $255.00 in the USA and Canada; £176.00 in Europe; £182.00 in Australasia and Japan; and £180.50 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).

AIX menus

A while ago, I developed a full-screen menuing system for VM/VSE (the system was presented in an article in *VSE Update* Issue 21). Given how useful this system has been to our installation, I decided to port it to an AIX environment, and have, therefore, written a similar system using a Korn shell script. An interesting point is that the **ksh** script uses much less code than its REXX equivalent, thanks to **ksh**'s **select menu_list** function.

The main reason for writing this menuing system is to provide a means of organizing commands and scripts in one place. End-users need only remember one command (**menu**) to invoke all the commands and scripts they need. This relieves them of the burden of memorizing commands and scripts, which, in a large installation, is quite onerous. This is especially true of the AIX environment, where commands often have a bewildering array of flags and operands.

This menuing system also makes it easier for system administrators to provide end-users with exactly what they require to access system functions. All supported scripts and commands can be controlled from one central location. Users themselves also have the opportunity to set up their own tables, by grouping commands they use frequently in a customizable table (*menuuser*) in their *$home/bin* directory.

The main script is called **menu**. End-users are presented with a menu system after entering the command **menu** at their workstation. Thereafter they simply pick a numeric selection and the system does the rest. All frequently used scripts and commands can be made available on the menus. Users navigate through menus and submenus, scrolling back and forth, in search of the required function. To re-display the menu, they simply press 'enter'.

You'll find that, with this system, the time required to train new users is reduced. This means that users become productive sooner than they would if they had to use a standard AIX system.

MENU is truly table-driven, which means that anyone can tailor it to their shop's specific needs by doing nothing more complex than creating a number of two-column tables, an example of which is shown below.

```
==> Exit to prompt          -exit@

AIX systems                 -MENU MENUAIX.tab@

CICS systems                -MENUCIC1@

SFS systems                 -MENUSFS1@

User-defined functions      -MENU MENUUSER.tab@

SMIT                        -smitty@
```

In this example (which corresponds to a file called *menu.tab*), the first item on each line is the menu item descriptor. The second is either the name of the submenu or a series of commands. A minus sign ('-') is used as the separator and an 'at' sign ('@') is the newline indicator. The '@' is chosen because of IFS override in the script. I include the prompt '== >' in all tables to show users the way out of the menuing system. An alternative to this is to use CTRL-C. As you can see, the menu script is itself invoked to create submenus – for instance, by the entry **menu menuaix.tab**. All table entries are sorted on column one before presentation.

While there is no limit on how many items you can have in your tables, if the list is too long, items will scroll off the screen. I recommend breaking long lists into separate tables, grouping commands and scripts by function.

As this menuing system is table driven, you should be able to implement it in your shop in less than an hour. Here are the steps necessary for a speedy implementation:

1 Download all of the **ksh** scripts and tables from Xephon's Web site.

2 Update *menu.tab* so that it includes references to all submenu tables. Make sure you include the two special characters '-' and '@' wherever necessary.

3 Set up other submenu tables with your scripts and commands. If necessary, enter multiple commands on one line by separating them with semicolons (';').

4 Change variable *header_message* to suit your company's needs in **menu** script.

5 Change variable *cics_naming_standard* to reflect your company's standard in the **menucic1** script.

6 Change variable *sfs_naming_standard* to reflect your company's standard in the **menusfs1** script.

7 Change variable *where* to your *bin* directory.

8 That's all! Issue the **menu** command, sit back, and enjoy!

MENU KSH

```
#!/bin/ksh
#
# This script sets up the full-screen selection feature
# accessing tables.
#
# Date        Author                      Description
# 970716      Dave Tang                   Program originator
#
# MENU ksh is the main script. It reads a MENU table in your
# $HOME/bin directory. This ksh script can also be called by
# other scripts that need to set up a selection menu. If so,
# you need to supply a table name as a parameter. There are
# two columns in supply table. The column to the left of the
# '-' is the command description for the user. The column to
# the right of the '-' is the actual command or another ksh
# script to be called. Use an '@' sign as a line terminator.
# You can nest as many tables as you wish.
#
# USAGE="usage: $0 TableName Text Text2"
#
# Tablename, Text, and Text2 are optional. Tablename if not
# supplied, the default is MENU in your $HOME/bin. Text is
# the message to override the standard header. Text2 is the
# selected target system. It can be an sfs server or a
# cics region.
#
# Other script and tables related to MENU
# MENU.tab       - Main menu table required to start MENU
# MENU.tmp       - work file
# MENUCIC1       - script to set up CICS environment
# MENUCIC1.tab   - set up by MENUCIC1 for region selection
# MENUCIC2.tab   - CICS selection table
# MENUCIC3.tab   - CICS selection table
# MENUSFS1        - script to set up SFS environment
# MENUSFS1.tab    - set up by MENUSFS1 for SFS selection
# MENUSFS2.tab    - SFS selection table
# MENUUSER.tab    - End-user's own selection table
# MENUSET         - script to prompt user for selection variable
# MENUAIX.tab     - AIX selection table
#
# trap 'print "Program $0 because of Cntl C"' INT

header_message='Welcome to Southam Magazine & Information Group'
where='/home/dtang/bin'

# if a table is supplied, use it, otherwise use $HOME/bin/MENU.tab
if [[ $# -eq 0 ]]
then
   table_name="$where/MENU.tab"
else
   table_name="$1"
fi
text=$2
text2=$3

# PS3 is the default prompt for select to set up the menu
PS3="Enter your choice, to redisplay menu, hit enter: "

# check to see if the table name exist.
if ! [[ -f $table_name ]]
then
  print "Cannot locate table file $table_name"
  print "Please correct and re-submit"
  exit
fi

# check whether the user table exists; if not, copy one from
# 'where'; otherwise, use the MENUUSER.tab the user has set up
if ! [[ -f $HOME/MENUUSER.tab ]]
then
  cp $WHERE/MENUUSER.tab $HOME/MENUUSER.tab
fi

table_sorted=$HOME/MENU.tmp         # all tables entries are sorted
sort -o $table_sorted $table_name

# read the table entries and concatenate them to build variable
while read line                     # read table entries
do
 list="$list $line"
done < $table_sorted
```

```
IFSsave=$IFS                        # save current IFS
IFS='@'                             # use @ as field separator
clear
uname -n | read machine

if [[ $text2 != '' ]]
then
   target_system=$text2
else
   target_system="N/A"
fi

function header
{
print -r '*'
print -r "* $header_message"
print -r '*'
print -r ''
print -r "       Accessing table => $table_name "
print -r "    machine, subsystem => $machine, $target_system"
print -r ''

if [[ $text != '' ]] && [[ $text != 'NONE' ]]
then
   print -r $text
else
   print -r '**** Description ***********  **** Commands **********'
fi
}
# end of header function

header
select menu_list in $list           # set up display menu
do
  command=${menu_list#*-}           # shift left until '-'
  IFS=$IFSsave                      # to allow operand in command
  case $command in                  # use command instead of menu_list
       exit)   clear
               exit;;
       *) eval $command
          print "\nCommand processed successfully";;
  esac
done
```

menuset is a script that sets environment variables passed to other scripts or subcommands. This is necessary as **ksh** does not allow a child script to pass a variable back to the parent script. Run **menuset** using a period ('.') to pass the variable.

MENUSET KSH

```
#!/bin/ksh
#
# ksh does not allow a child script to pass parameters back to
# its parent script. To overcome this problem, MENUSET is coded
# to pass variables to a subsequent script or command. In our
# case, this is a CICS region name or an SFS server name.
#
# Date        Author                      Description
# 970512      Dave Tang                   Program originator
#
# USAGE="usage: $0"
#

stop='N'
# Set up a 'while loop' to ask for the inquiry
while [[ $stop = 'N' ]]
do
  print ""
  print "Please enter the data which you want to inquire"
  read input_parm
  if [[ $input_parm = '' ]]
  then
     print "You did not provide any input"
  else
     stop='Y'
     export MENUINFO=$input_parm
  fi
done
```

MENUCIC1

```
#!/bin/ksh
#
# MENUCIC1 - This script asks the user to select a CICS region.
# It calls the MENU main script to set up a selection menu.
#
# MENUCIC1.tab - the CICS region selection table
# MENUCIC2.tab - the CICS function selection table
# MENUCIC3.tab - the CICS resource inquiry selection table
#
# Date        Author                      Description
# 970512      Dave Tang                   Program originator
#
# USAGE="usage: $0"

cics_naming_standard="cics.SSCICS"  # your CICS naming standard

work_file=$HOME/bin/MENU.tmp        # set up temp work file
```

```
file_name=$HOME/bin/MENUCIC1.tab

lssrc -a | grep $cics_naming_standard > $work_file
if [[ -e $file_name ]]            # if work file exist
then
   rm $file_name
fi

while read line                   # build CICS region table
do
  cics_region=${line%% *}         # keep only the 1st variable
  cics_region=${cics_region#*.}   # keep only the region name
  print "$cics_region                  -print $cics_region >
$work_file;exit@" >> $file_name
done < $work_file

# text_info has to be in one continuous line for now as a
# positional parm.
text_info='Please.select.your.desired.CICS.region'

# call MENU script with the table name and text info
MENU MENUCIC1.tab $text_info       # display selection menu

read cics_region < $work_file
text_info='NONE'
export REGION=$cics_region
MENU MENUCIC2.tab $text_info $cics_region
```

MENUCIC1.TAB (CREATED BY MENUCIC1)

```
SSCICS1                -print SSCICS1 > /home/dtang/bin/MENU.tmp;exit@
SSCICS2                -print SSCICS2 > /home/dtang/bin/MENU.tmp;exit@
SSCICS3                -print SSCICS3 > /home/dtang/bin/MENU.tmp;exit@
SSCICS4                -print SSCICS4 > /home/dtang/bin/MENU.tmp;exit@
SSCICS5                -print SSCICS5 > /home/dtang/bin/MENU.tmp;exit@
SSCICS6                -print SSCICS6 > /home/dtang/bin/MENU.tmp;exit@
```

MENUCIC2.TAB

```
==> Return Previous    -exit@
View console.msg       -view /var/cics_regions/$REGION/console.msg@
View console.msg1      -view /var/cics_regions/$REGION/console.msg.1@
View console.msg2      -view /var/cics_regions/$REGION/console.msg.2@
View all console.msg*  -view /var/cics_regions/$REGION/cons*@
View CSMT log          -view /var/cics_regions/$REGION/data/CSMT.out@
View CSMT log1         -view /var/cics_regions/$REGION/data/CSMT.out.1@
View CSMT log2         -view /var/cics_regions/$REGION/data/CSMT.out.2@
View all CSMT logs     -view /var/cics_regions/$REGION/data/CSMT*@
```

```
Inquire CICS resources -MENU MENUCIC3.tab@
View symrec            -view /var/cics_regions/$REGION/symrecs@
Display CICS CPU usage -ps caugx | grep cics | head -12@
```

MENUCIC3.TAB (INQUIRES ABOUT CICS RESOURCE DEFINITION)

```
Communication              -cicsget -c cd  -r $REGION -l | more@
Transaction                -cicsget -c td  -r $REGION -l | more@
Files                      -cicsget -c fd  -r $REGION -l | more@
Gateways                   -cicsget -c gd  -r $REGION -l | more@
Gateway Servers            -cicsget -c gsd -r $REGION -l | more@
Journals                   -cicsget -c jd  -r $REGION -l | more@
Listeners                  -cicsget -c ld  -r $REGION -l | more@
Monitoring                 -cicsget -c md  -r $REGION -l | more@
Programs                   -cicsget -c pd  -r $REGION -l | more@
Regions                    -cicsget -c rd  -r $REGION -l | more@
Schema File Definitions    -cicsget -c scd -r $REGION -l | more@
Transient Data Queues      -cicsget -c td  -r $REGION -l | more@
Temporary Storage Queues   -cicsget -c tsd -r $REGION -l | more@
Users                      -cicsget -c ud  -r $REGION -l | more@
Terminals                  -cicsget -c wd  -r $REGION -l | more@
Products                   -cicsget -c xa  -r $REGION -l | more@
==> Return Previous        -exit@
```

MENUSFS1 (SCRIPT TO SET UP SFS SERVER)

```
#!/bin/ksh
# MENUSFS1 - This script asks the user to select an SFS server.
# It calls the MENU main script to set up a selection menu.
#
# MENUSFS1.tab - the SFS server selection table
# MENUSFS2.tab - the SFS server function selection table
#
# Date        Author                     Description
# 970515      Dave Tang                  Program originator
#
# USAGE="usage: $0"

sfs_naming_standard="SFS"           # your CICS naming standard

work_file=$HOME/bin/MENU.tmp        # set up temp work file
file_name=$HOME/bin/MENUSFS1.tab
sfs_directory="/var/cics_servers/SSD/cics/sfs"
ls $sfs_directory | grep $sfs_naming_standard > $work_file
if [[ -e $file_name ]]              # if work file exist
then
   rm $file_name
fi
```

```
while read line                    # build sfs server table
do
  print "$line                  -print $line > $work_file;exit@" >>
$file_name
done < $work_file

text_info='Please.select.your.desire.SFS.region'
MENU MENUSFS1.tab $text_info        # display selection menu

read sfs_region < $work_file
text_info='NONE'
export REGION=$sfs_region
export ENCINA_SFS_SERVER="/.:/cics/sfs/$REGION"
export ENCINA_TK_SERVER=$ENCINA_SFS_SERVER
MENU MENUSFS2.tab $text_info $sfs_region
```

MENUSFS1.TAB (SET UP BY MENUSFS)

```
SFSCICS1            -print SFSCICS1 > /home/dtang/bin/MENU.tmp;exit@
SFSCICS2            -print SFSCICS2 > /home/dtang/bin/MENU.tmp;exit@
SFSCICS3            -print SFSCICS3 > /home/dtang/bin/MENU.tmp;exit@
SFSCICS4            -print SFSCICS4 > /home/dtang/bin/MENU.tmp;exit@
SFSCICS5            -print SFSCICS5 > /home/dtang/bin/MENU.tmp;exit@
SFSCICS6            -print SFSCICS6 > /home/dtang/bin/MENU.tmp;exit@
```

MENUSFS2.TAB

```
==> Return Previous    -exit@
List logical volumes   -sfsadmin list lvols@
List files (all)       -sfsadmin list files | more@
Query file by name     -. MENUSET;sfsadmin query file $MENUINFO | more@
List files (OFD)       -sfsadmin list ofds | more@
Close file by id       -. MENUSET;sfsadmin terminate ofd $MENUINFO@
Query SFS usage        -. MENUSET;sfsadmin query lvol $MENUINFO | more@
Query server inf       -sfsadmin query server@
Query working dir      -tkadmin query workingdirectory@
List transaction       -tkadmin list transactions@
Query checkpoint intvl -tkadmin query checkpointinterval@
Query identity         -tkadmin query identity@
```

MENUUSER.TAB (DEFAULT END-USER TABLE)

```
==> Return Previous       -exit@
Who is logged on?         -whoami@
Display env. variables    -env | more@
Display printer status    -lpstat@
What is the OS level      -oslevel@
Change my password        -passwd@
```

```
What is my current dir.   -pwd@
What machine I logged on? -uname -n@
Check backup log          -/home/iopsc02/seelog.sh@
```

MENUAIX.TAB

```
==> Return Previous       -exit@
Query File System         -df | more@
Query Env variables       -env | more@
Query login               -finger | more@
Change password           -passwd@
Query Alias               -alias | more@
Query Printers            -lpstat | more@
Query Devices             -lsdev -C -H | more@
Query Memory              -lsattr -El sys0 -a realmem@
Query Physical Volumes    -lspv | more@
Query Paging usage        -lsps -a@
Query Subsystem           -lssrc -a | more@
Display IO stats          -iostat | more@
Display network usage     -netstat | more@
Display system stats      -vmstat@
Display current processes -ps -ef | more@
Display processes by name -. MENUSET;ps -ef | grep -i $MENUINFO | more@
Display top 10 CPU users  -ps gvaxc | head -10@
```

Dave Tang
Manager, Systems Engineering
Southam Information (Canada)

Serial Storage Architecture versus SCSI

The growth in power of CPUs over the past two decades has been substantial, allowing processors to keep up with the demands made on them by an ever increasing volume of data that needs processing. The performance, and more importantly price/performance, of today’s Unix and NT servers doubles every 12 to 18 months. Today’s Intel desktop provides over 100 times the performance of the original IBM PC, for close to the original price.

However, the technology for storing, managing, and accessing data has not kept up with the growth in demand for data storage. Today many companies have storage strategies for midrange and small data servers based on the SCSI protocol, which has increased its maximum performance by only a factor of eight in the same time as processor performance has grown by a factor of over 50.

The industry has been waiting for the arrival of a storage technology that provides users with significant performance, capacity, and reliability benefits, while at the same time lowering the overall cost of storage. IBM's Serial Storage Architecture, SSA, has brought this and more to the industry. For an introduction to SSA and its advantages, see *SSA – Serial Storage Architecture* in *AIX Update* Issue 11, September 1996.

THE SCSI STANDARD

The Small Computer Systems Interface (SCSI) became the *de facto* standard for midrange systems, especially those running Unix, in the 1980s. While the performance and throughput of SCSI has been enhanced over the past ten years, the underlying technology and protocols have remained essentially unchanged.

The bandwidth of the original SCSI-1 standard was 5 Mb/sec transmitting 8-bit words. The SCSI-2, SCSI-2 Fast, SCSI-2 Fast/Wide, and Ultra SCSI have significantly enhanced the original standard, increasing the word size to 16-bits and extending the maximum data transfer rate to 40 Mb/sec.

SCSI ISSUES AND LIMITATIONS

SCSI is a bus-based architecture, meaning that all communication between SCSI devices and the host system are over a 68-pin cable (for SCSI-2 Differential Fast/Wide). The maximum total length of cable from the CPU to the terminator is 25 meters. This effectively rules out remotely connected SCSI devices. SCSI cabling is not only cumbersome, but also adds to the cost of SCSI-based systems. You can have a total of seven SCSI devices on a SCSI interface adapter any time. Ultra SCSI increases this to a total of 15 devices on an adapter card. It takes coordination to transfer information between devices or between a computer and a device successfully – a process known as arbitration.

The main limitation of SCSI interfaces is generally not a result of shortcomings in data throughput. Instead, it's usually the result of problems with access to data via the shared arbitrated bus that connects the SCSI devices. While the speed and width of the SCSI bus have increased over time, when there are multiple devices on a shared bus, and more than one is attempting to transfer data, performance still suffers. In many cases, the communication overhead arising from arbitration between devices attempting to transfer data is larger than the actual volume of data being transferred.

Data flow across the SCSI bus can be compared to a single lane bridge that is used to transfer information to and from both sides. Information can only flow from a single source, in a single direction. Arbitration manages traffic across the bridge. However, the more arbitration, the less data can be transferred.

THE NEW STANDARDS

Recognizing that the performance of disk subsystems is becoming more of an issue, vendors have been working diligently to introduce storage technology that has the ability to meet the scalability and reliability requirements of the future. Different technologies have been proposed to alleviate the bottlenecks in data transfer rates and other limitations of SCSI.

The new technologies centre around creating serial interfaces that extend distances and increase data transfer rates between devices. The concept of serial interfaces between subsystems brings several benefits to storage subsystems. A comparison of the characteristics of typical serial interfaces and the SCSI bus is shown below.

	Serial interface	SCSI bus
Maximum number of devices that can be attached	Over 120	Up to 15 (7)

	Serial interface	SCSI bus
Maximum distance between attached devices	Over 25 metres	Up to 3 meters
Type of cabling	Serial wire or fibre-optic	68-pin shielded
Cabling costs	Inexpensive	Fairly expensive
Bandwidth	Up to 100 Gb/sec	Up to 40 Mb/sec
Maximum number of hosts supported	Up to 8	Up to 2
Maximum distance between storage devices and host	Up to 2500 meters	A few meters

Two standards for serial interfaces are currently being promoted in the industry. One is an arbitrated serial loop called FC-AL (Fibre Channel Arbitrated Loop). The second is a non-arbitrated serial loop, called SSA (Serial Storage Architecture).

FIBER CHANNEL ARBITRATED LOOP

FC-AL uses an ‘arbitrated loop’, meaning that there remains a single-lane bridge for data flow. Its 100MB/sec data transfer rate and support of up to 126 devices on a loop are much higher than SCSI. Several vendors plan to deliver FC-AL products to the market, though none are shipping at the time of writing.

Issues concerning FC-AL

- The FC-AL protocol incurs an overhead for each data transfer associated with loop arbitration.
- Frames (units of information transfer) are transmitted in only one direction around the entire loop.
- Only two devices can communicate at any time on the loop.
- If a device is removed, or the link fails, the entire subsystem fails. The only way to avoid this is to install an optional, redundant second loop.

- All devices in the loop share in the work of transmitting data, which passes through each node in the loop.

Although FC-AL is a new design that offers substantially higher performance than SCSI, FC-AL still suffers from many of the I/O bottlenecks of SCSI, and is more expensive than SCSI.

SSA

A second serial access technology is IBM's Serial Storage Architecture. SSA is a low-cost, high performance serial interface that was designed not only for disks, but also for other I/O devices such as CD-ROM drives, optical drives, tape drives, and even printers.

Although the SSA specification allows the connection of up to 128 disk drives, current SSA adapters from IBM support only up to 48 disks on a loop and two loops per adapter. Each loop contains two read paths and two write paths, all of which are able to transfer data concurrently at up to 20 Mb/sec.

SSA adapters can transfer data on each loop's four data paths simultaneously; each data path runs at 20 Mb/sec, so the effective bandwidth is 80 Mb/sec. Given that each adapter supports two loops, adapters provide an effective bandwidth of 160 Mb/sec.

SSA cabling costs are significantly lower than SCSI. SSA requires a simple dual twisted pair cable compared with SCSI-2's 68-pin cable. SSA devices may be located up to 82 feet apart, and IBM has introduced a Fibre-Optic Extender that extends the maximum distance between nodes to 2400 meters.

For both SCSI and FC-AL, the arbitration overhead is the limiting factor to data transfer rates. SSA overcomes this using an un-arbitrated serial loop. Consequently, initiating a data transfer on an SSA loop requires only three commands, compared with 12 that are required to initiate a transfer on a SCSI bus (eight of the 12 commands are required solely for arbitration).

SSA adapters select the optimum data path between devices in order to maximize performance. If a link fails, the SSA adapter simply chooses an alternative route, rather than crippling communication to the entire drive. Hot-swappable cables and drive modules on SSA disk subsystems add to its high-availability features. Below is a comparison of the main features of SSA subsystems and their SCSI counterparts.

Feature	SCSI	SSA
Cable type	68-conductor shielded cable (SCSI-2 Differential, Fast/Wide)	4-conductor twisted-pair
Maximum cable length	25 metres total	25 metres between devices
Cable costs	High	Low
Device addressing	Set manually	Set dynamically
Cable termination	Terminator required at end of bus	Not required (closed serial loop)
Fault tolerance	Only one data path (bus) is provided; if it fails, communication with the device is lost	No single point of failure (multiple data paths)
Maximum throughput	Up to 40Mb/sec	Up to 160 Mb/sec
Disk drives supported	SCSI	SSA and SCSI compatible

SUMMARY

Meeting future storage requirements means that disk subsystems will have to be larger, faster, more reliable, more efficient, and have better price/performance than ever before. Serial technologies, such as IBM’s SSA, represent a shift in storage technology for small to medium-sized systems that may satisfy future needs for storage and, in time, replace the popular SCSI standard.

Werner Klauser
Klauser Informatik (Switzerland)

Monitoring AIX with PCs revisited

In a previous article in *AIX Update* (see *Monitoring AIX with PCs*, Issue 22 pages 3–32 and Issue 23 pages 12–38) I presented a set of tools for monitoring AIX using PCs. The original tools provided facilities for real-time monitoring and reporting of AIX systems, and were primarily for monitoring filesystem and CPU utilization. This is now expanded to system-wide monitoring. The new tools presented in this article allow you to focus on what is important in your system, thus making the production of system reports more streamlined and enabling you to analyse performance problems in a way that is more meaningful to your installation.

The tools generate both graphs and reports, thus doing more than just putting figures on a page. In my experience, managers like graphs, even though every one seems to have his or her own preferrence as to style and presentation.

The idea behind the tools is to collect AIX system statistics in text files that PC spreadsheets and database packages can process. I'll start by dealing with **sar** reports, turning them into comma-separated text files. This is, obviously, not an option if you don't have **sar** reporting set up, so I'll also discuss ways of monitoring and reporting using standard system monitoring tools. While regular **sar** files contain much useful information, their style of presentation can make them difficult to use with PCs. By turning **sar** files into CSV text files that PCs can read without cutting and pasting, the process of monitoring and reporting is made easier and less prone to error.

A sample **sar** output is shown in Figure 1 opposite. It shows the output of **sar -u**, which reports on CPU activity. On some PC packages, such output can be loaded, without modification, as a space-delimited file. This is not an ideal solution as it requires data to be in columns and may affect the formatting of the -resulting data. Also, analysis of data spanning more than one day requires the data to be manipulated further.

The example that follows assumes that **sar** is set up to produce a file for each day of the month and that, at any time, only one month's worth of statistics exists (files with older data are overwritten). **sar** files are placed in the directory */var/adm/sa/* and have names conforming to the pattern *sadd*, where the *dd* is the day.

The format of this file can easily be turned into comma separated text using the following **sed** command (which replaces one or more spaces with a comma):

```
sar -f /var/adm/sa/sadd | sed "s/[ ]*[ ]/,/g" > textfile.txt
```

This format is more useful than the previous one as it separates data items with commas, though it still leaves headers and footers in the output. These can cause problems to PC packages when the file is loaded, so it's best to eliminate them. However, they also contain useful information, such as the date of the measurements, that would be lost if they're simply deleted. Therefore they need to be processed, for instance, by encoding the date in each entry in the file.

If the file is to be used by a relational database, then a further requirement is that it should have a 'key' field. This has the added benefit of making the data much easier to use. A key field enables easy loading and appending of files as it eliminates duplicate records within the database. This would, for instance, prevent the same file being loaded into the database twice. A suitable candidate for a key field is the date (including the time), which is unique within a **sar** collection.

The process discussed below uses **sar** files that already exist, making

```
AIX serv1 1 4 002019375900    10/09/97

00:00:02    %usr    %sys    %wio   %idle
01:00:01       5       15    2       79
02:00:01       5       13    2       81
..
..
22:00:02      58      19     3       20
23:00:02      29      17     2       51

Average       38      18     2       42
```

Figure 1: Sample ***sar*** *report*

it necessary to be able to process a previous day's output. This involves being able to generate a previous date for the key field. Once the file name is generated, the existing data has to be processed into a comma separated file. At the end of the process any necessary housekeeping takes place. The entire process can be broken down into the following five steps:

1 Generating the name of the **sar** file.

2 Generating delimited text files.

3 Scheduling the process.

4 Transferring the files.

5 Housekeeping.

GENERATING THE SAR FILE NAME

The first step is to generate the date for which data is to be processed. This enables the system to select the correct **sar** file and to generate (if necessary) a key field for later use. The script below (**calcdate**) calculates dates back to the start of the previous year using a given day parameter. The output of this script is used to calculate the key field, select the correct **sar** file, and convert the date to a format suitable for use by **sed** (used later in the controlling scripts).

calcdate may have up to eight parameters passed to it, though, for the purpose of this example, it needs to be in the format shown below, which also appears in the **format** script.

```
calcdate 1 -s -yj -d -sp
```

The first parameter specifies the number of days previous to the current date for which the date is to be calculated. For instance:

1 Specifies that the previous day's date is to be calculated.

3 Specifies that the date from three days previously is to be calculated (this is useful for calculating Friday's date on Mondays).

The second parameter specifies the separator for the output. In this instance the parameter is:

-s Fields in the output are to be separated by spaces.

The third, fourth, and fifth parameters specify the date format for the output.

-yj Specifies that the output should comprise the year (*yy*) and Julian date.

-d Specifies that the output should comprise the day number in the month (used to get the **sar** file)

-sp Specifies that the date should be formatted for use by **sed** (in *dd\/mm\/yy* format).

CALCDATE SCRIPT

```
#!/bin/ksh
integer gto
integer DATEJ
integer YEAR
integer TOTAL_DAYS
integer LEAP_DAYS
integer NUM_WEEKS
integer CALC_WEEKS
integer DIFF_DAYS
integer DAY_INT
integer jul
integer year
integer lyear
integer diff
integer LEAR_TOT
MINUS=$1
filedat[1]=31
filedat[2]=28
filedat[3]=31
filedat[4]=30
filedat[5]=31
filedat[6]=30
filedat[7]=31
filedat[8]=31
filedat[9]=30
filedat[10]=31
filedat[11]=30
filedat[12]=31
PASS[1]=$1
PASS[2]=$2
PASS[3]=$3
PASS[4]=$4
PASS[5]=$5
PASS[6]=$6
```

```
PASS[7]=$7
PASS[8]=$8
SET="N"
leap_year_valid()
{
integer r
r=$1
let FOUR=$r/4*4
let HUNDRED=$r/100*100
let FOUR_HUN=$r/400*400
if [[ "$FOUR" = "$r" && "$HUNDRED" != "$r" || "$FOUR_HUN" = "$r" ]]
then
LEAP_TOT=366
SET="Y"
return 0
else
LEAP_TOT=365
SET="N"
return 1
fi
}
name_of_day ()
{
#Calculate day name (taking Saturday as the first day and Sunday
#as day zero for decimal display)
DATEJ=$jul
YEAR=$year
if [ "$SET" = "Y" ]
then
    LEAP_DAYS=YEAR/4-1
else
    LEAP_DAYS=YEAR/4
fi
TOTAL_DAYS=$YEAR*365+LEAP_DAYS+DATEJ
NUM_WEEKS=$TOTAL_DAYS/7
CALC_DAYS=$NUM_WEEKS*7
DIFF_DAYS=$TOTAL_DAYS-$CALC_DAYS
case "$DIFF_DAYS" in
    2)
        DAYNAME="Sunday"
        DAY_INT=0
    ;;
    3)
        DAYNAME="Monday"
        DAY_INT=1
    ;;
    4)
        DAYNAME="Tuesday"
        DAY_INT=2
    ;;
    5)
        DAYNAME="Wednesday"
        DAY_INT=3
    ;;
    6)
        DAYNAME="Thursday"
        DAY_INT=4
    ;;
    0)
        DAYNAME="Friday"
        DAY_INT=5
    ;;
    1)
        DAYNAME="Saturday"
        DAY_INT=6
    ;;
    esac
}
generate_year_day()
{
year=$(date +%Y)
jul=$(date +%j)-$MINUS
leap_year_valid $year
DAY_LOOP=$jul
while [ $DAY_LOOP -le 0 ]
do
    year=$year-1
    leap_year_valid $year
    let DAY_LOOP=DAY_LOOP+$LEAP_TOT
done
let jul=$DAY_LOOP
}
M_D_Y_calc()
{
leap_year_valid $year
if [[ "$?" = "0" ]]
then
    let filedat[2]=29
else
    let filedat[2]=28
fi
leapjul=$jul
let i=1
let total=0
while [[ i -lt 13 ]]
do
    if [[ $total -lt $jul ]]
    then
        temp=${filedat[i]}
        let total=$total+$temp
        let month=$i
    fi
    let i=i+1
done
let temp=${filedat[month]}
let str=$total-$temp
let day=$jul-$str
amonth="$month"
aday="$day"
if [[ month -lt 10 ]]
then
    amonth="0$month"
fi
if [[ day -lt 10 ]]
then
    aday="0$day"
fi
}
month_name()
{
case "$amonth" in
    01)
        LONGMON="January"
    ;;
    02)
        LONGMON="February"
    ;;
    03)
        LONGMON="March"
    ;;
    04)
        LONGMON="April"
    ;;
    05)
        LONGMON="May"
    ;;
    06)
        LONGMON="June"
    ;;
    07)
        LONGMON="July"
    ;;
    08)
        LONGMON="August"
    ;;
    09)
        LONGMON="September"
    ;;
    10)
        LONGMON="October"
    ;;
    11)
        LONGMON="November"
    ;;
    12)
        LONGMON="December"
    ;;
    esac
}
if [ "$1" = "?" ]
then
    print " Options available"
    print " ----------------"
    print " -n      Day Number (0-6 0=Sunday)"
    print " -nn     Short Day Name eg Mon"
    print " -nnn    Long Day Name eg Monday"
    print " -d      Day Of Month (With leading 0)"
    print " -m      Month Number (With leading 0)"
    print " -mm     Short Month eg Jan"
    print " -mmm    Long month name eg January"
    print " -y      Short Year eg 97"
    print " -yy     Long Year eg 1997"
    print " -j      Julian Day"
    print " -yj     Year and julian day eg 97261"
    print " -sp     Output mm\/dd\/yy"
    exit 0
fi
########Check number of paramenters passed, exit if too many
if [ $# -gt 8 ]
then
    print "Too many parameters"
    exit
fi
generate_year_day
M_D_Y_calc
name_of_day
month_name
SPACE=""
gto=1
while [[ gto -lt 8 ]]
do
gto=gto+1
if [ "${PASS[$gto]}" = "-n" ]
then
    OUT_PASS[$gto]=$DAY_INT
fi
if [ "${PASS[$gto]}" = "-nn" ]
then
    OUT_PASS[$gto]=`echo $DAYNAME|cut -c1-3`
fi
```

```
if [ "${PASS[$gto]}" = "-nnn" ]
then
    OUT_PASS[$gto]=$DAYNAME
fi
if [ "${PASS[$gto]}" = "-d" ]
then
    OUT_PASS[$gto]=$aday
fi
if [ "${PASS[$gto]}" = "-m" ]
then
    OUT_PASS[$gto]=$amonth
fi
if [ "${PASS[$gto]}" = "-mm" ]
then
    OUT_PASS[$gto]=`echo $LONGMON|cut -c1-3`
fi
if [ "${PASS[$gto]}" = "-mmm" ]
then
    OUT_PASS[$gto]=$LONGMON
fi
if [ "${PASS[$gto]}" = "-y" ]
then
    OUT_PASS[$gto]=`echo $year|cut -c3-4`
fi
if [ "${PASS[$gto]}" = "-yy" ]
then
    OUT_PASS[$gto]=$year
fi
if [ "${PASS[$gto]}" = "-j" ]
then
    OUT_PASS[$gto]=$jul
fi
if [ "${PASS[$gto]}" = "-yj" ]
then
    OUT_PASS[$gto]=`echo ${year}$jul|cut -c3-`
fi
if [ "${PASS[$gto]}" = "-sp" ]
then
    OUT_PASS[$gto]=$aday"\/"$amonth"\/"$year
fi
if [ "${PASS[$gto]}" = "-s" ]
then
    SPACE=" "
fi
if [ "${PASS[$gto]}" = "-/" ]
then
    SPACE="/"
fi
done
OUTPUT=""
```

```
gto=1
while [[ gto -lt 8 ]]
do
    gto=gto+1
    if [ "${OUT_PASS[$gto]}" != "" ]
    then
        addt=gto+1
        if [ "${OUT_PASS[$addt]}" != "" ]
        then
            OUTPUT=$OUTPUT"${OUT_PASS[$gto]}"$SPACE
        else
            OUTPUT=$OUTPUT"${OUT_PASS[$gto]}"
        fi
    fi
done
print $OUTPUT
#end
```

GENERATING THE DELIMITED TEXT FILES

Each **sar** file and all its parameters need to be processed separately, the end result being a file that PC packages can format and process correctly. The scripts in this article support all the **sar** parameters below for system-wide reporting. They do not, however, break down information in reports to individual processors using the **-P** flag. If you require this level of reporting, then you should modify the scripts.

-*a*	Reports on the use of file access system routines.
-*b*	Reports on buffer activity.
-*c*	Reports on system calls.
-*k*	Reports on kernel process activity.
-*m*	Reports on message and semaphore activity.
-*q*	Reports queue statistics.
-*r*	Reports paging statistics.
-*u*	Reports processor statistics.
-*v*	Reports on process, kernel-thread, i-node, and file table status.
-*w*	Reports on system switching activity.
-*y*	Reports on **tty** device activity.

The idea is to process data into delimited text files for use by PC packages. In our set-up, four different types of delimited file are generated. Two them comprise collected **sar** statistics for particular calendar months. One file contains raw data and the other averaged data. This provides us with historical data for our month-end reporting, also enabling additional month-end reporting to be carried out quickly. The other two sets of files are generated and stored for specifically requested days; these files are loaded into a database and are available for historical or trend analysis of system performance. Again, raw data and averages are kept in separate files. This enables a more flexible approach, though you may wish to modify the scripts if you don't require this level of flexibility.

The next script (**control_load**, shown below) controls the entire process. It checks the requested number of days' worth of **sar** files to be processed and moves the formatted files to the correct place on the system. The script requires three (positional) parameters that need to be in the format shown below.

```
control_load 1 -n -zz
```

The first parameter is the number of previous days' worth of data to be processed.

1	Only the previous day's **sar** file is to be processed.
3	The **sar** file from three days previously is to be processed (for instance, if called on Monday, then Friday's file is processed).

The second parameter controls the process of putting a header in the day files.

-n	No header is placed in the files for the day.
-h	A header is placed in the files for the day.

The third parameter controls which **sar** parameter options are to be processed.

-zz	Results in all **sar** parameter options being processed.
"-a -c"	Only file access system routines and system calls are processed.

CONTROL_LOAD SCRIPT

```
#!/bin/ksh

#Variable Definition
HOME=/usr/home/it032x
DATA_HOME=$HOME/system_stats
MONTH_FILE=$DATA_HOME/month_files
MACHINE_ID="MAN"

delete_files()
{
for d in $SAR_STRING
do
    SEL=`echo $d|cut -c 2|tr -A "[a-z]" "[A-Z]"`
    rm -f $DATA_HOME/${MACHINE_ID}???${SEL}.TXT
done
rm -f $DATA_HOME/HEDSAR*
}

month_file_control()
{
FILE_TAG_LONG=`$HOME/calcdate $LOOP -s -mm -d|tr "[a-z]" "[A-Z]"`
FILE_TAG=`echo $FILE_TAG_LONG|awk '{print $1}'`
DAY_TAG=`echo $FILE_TAG_LONG|awk '{print $2}'`

if [[ "$1" = "SAR" ]]
then
    DATA_CONTAIN=${DATA_HOME}/Datafile3
    DATA_OUTPUT=${MONTH_FILE}/${MACHINE_ID}${SEL}${FILE_TAG}.TXT
    HEAD=${DATA_HOME}/HEDSAR
else
    DATA_CONTAIN=${DATA_HOME}/Datafile4
    DATA_OUTPUT=${MONTH_FILE}/${MACHINE_ID}${SEL}${FILE_TAG}A.TXT
    HEAD=${DATA_HOME}/HEDSARAVE
fi

grep `tail -1 $DATA_CONTAIN` $DATA_OUTPUT 1>/dev/null 2>&1

if [[ "$?" != "0" ]]
    then
    if [[ ! -s $DATA_OUTPUT ]]
    then
        cat $HEAD >> $DATA_OUTPUT
    fi
    cat $DATA_CONTAIN >> $DATA_OUTPUT
fi
}

daily_file_control()
```

```
{
if [[ "$1" = "SAR" ]]
then
    DATA_CONTAIN=${DATA_HOME}/Datafile3
    DATA_OUTPUT=${DATA_HOME}/${MACHINE_ID}SAR${SEL}.TXT
    HEAD=${DATA_HOME}/HEDSAR
else
    DATA_CONTAIN=${DATA_HOME}/Datafile4
    DATA_OUTPUT=${DATA_HOME}/${MACHINE_ID}AVE${SEL}.TXT
    HEAD=${DATA_HOME}/HEDSARAVE
fi
    if [[ ! -s $DATA_OUTPUT && "$HEADER" = "-h" ]]
        then
            cat $HEAD >> $DATA_OUTPUT
        fi

    cat $DATA_CONTAIN >> $DATA_OUTPUT
}

#Main Script Start

SAR_STRING=$3
HEADER=$2

if [[ "$SAR_STRING" = "-zz" ]]
then
    SAR_STRING="-a -b -c -k -m -q -r -u -v -w -y"
fi

delete_files

for i in $SAR_STRING

do
LOOP=$1

while [ LOOP -gt 0 ]
do
    print " $i $LOOP started `date +"%H:%M"`"

    $HOME/format $LOOP $i $2

    SEL=`echo $i|cut -c 2|tr -A "[a-z]" "[A-Z]"`

    #daily_file_control SAR

    month_file_control SAR

    if [[ "$i" != "-v" ]]
    then
    #    daily_file_control AVE
        month_file_control AVE
    fi

    print " $i $LOOP completed `date +"%H:%M"`"

    let LOOP=LOOP-1

done

rm -f $DATA_HOME/HEDSAR*
done
#end
```

The lines underlined in the script control the location where files generated by the script are kept and the RS/6000 machine name. They are as follows:

```
HOME=/usr/home/it032x
DATA_HOME=$HOME/system_stats
MONTH_FILE=$DATA_HOME/month_files
MACHINE_ID="MAN"
```

- *HOME* points to where script files reside.
- *DATA_HOME* points to where daily delimited text files reside.
- *MONTH_FILE* points to where monthly delimited text files reside.
- *MACHINE_ID* stores the RS/6000 system name for which statistics are being collected.

Make the necessary changes to these lines to suit your directory structure (don't forget to create the directories before running any scripts!).

The **control_load** script calls the **format** script to perform the necessary formatting of the files. This script strips out all the unnecessary parts of the normal **sar** output to produce a usable text file. The first procedure of the script is to remove the header and footer from the **sar** file. It does this simply by removing the top four lines (the header) and the last two lines (the footer). The fourth line of any **sar** report contains information about what is being collected, so this is saved to a separate file for later use. The line containing averages is also hived off into another file for later use. What remains is used for in-depth daily analysis.

The exception to the above is when processing output from **sar -v** (system tables), which contains no averaged information. In this instance only the last line of output (the footer) is stripped out and no files of averages are generated. The output of **sar -q**, which reports on queue statistics, also needs special handling. When this report is generated by **sar**, a zero value is represented by a null field in the report. Therefore, when formatting this output, eight spaces are hard coded for each of the four possible entries. Make sure you check the files generated from the output of **sar -q** to ensure they are consistent with the information held in the **sar** file.

The **format** script is called from **control_load** as shown below:

```
format $LOOP $i $2
```

The first positional parameter is the number of days previously for which data is to be processed.

$LOOP	The first positional parameter comes from the call to the **control_load** script.

The second parameter is the current **sar** parameter to process.

$i	Used to loop through the **sar** parameters.

The third parameter controls what is done with the header.

$2	The second positional parameter from the call to the **control_load** script.

FORMAT SCRIPT

```
#!/bin/ksh

HOME=/usr/home/it032x
DATA_HOME=/usr/home/it032x/system_stats
FORMLOG=${DATA_HOME}/form1
SAR_REPORTS=/var/adm/sa
VAR_SPLIT=`$HOME/calcdate $1 -s -yj -d -sp`
YEAR_JUL=`echo $VAR_SPLIT|awk '{print $1}'`
SAR_DATE=`echo $VAR_SPLIT|awk '{print $2}'`
YES_DATE=`echo $VAR_SPLIT|awk '{print $3}'`
PASSED=$2
```

```
>$DATA_HOME/Datafile2
>$DATA_HOME/Datafile4

header_line()
{
    HEAD_ROW=`cat $FORMLOG|head -4 |tail -1 | sed "s/[ ]*[ ]/,/g" |cut -
f2- -d,`
    print "Key,Date,Time,"$HEAD_ROW >$DATA_HOME/HEDSAR
    if [[ "$PASSED" != "-v" ]]
    then
    print "Key,Date,"$HEAD_ROW >$DATA_HOME/HEDSARAVE
    fi
}
queue_set()
{
    FIRST=`tail -$ROTATE $DATA_HOME/Datafile1 | head -1 |cut -c9-40|sed
"s/[ ][ ][ ][ ][ ][ ][ ][ ]/,/g"|sed "s/ * /,/g"`
    SECON=`tail -$ROTATE $DATA_HOME/Datafile1 | head -1 |cut -c1-8`
    echo $SECON$FIRST|sed "s/^/$KEY\,$YES_DATE\,/g">>$DATA_HOME/
Datafile2
}

sar $2 -f $SAR_REPORTS/sa${SAR_DATE} > $FORMLOG

if [[ ! -s $DATA_HOME/HEDSAR ]]
then
        header_line
fi

NO_LINES=`cat ${FORMLOG}|wc -l`

let TAIL_CHOP=NO_LINES-4
if [[ "$PASSED" != "-v" ]]
then
    let HEAD_CHOP=TAIL_CHOP-2
else
    let HEAD_CHOP=TAIL_CHOP-1
fi


cat ${FORMLOG}|tail -$TAIL_CHOP|head -$HEAD_CHOP>$DATA_HOME/Datafile1

COUNT=0

while [ $COUNT -lt $HEAD_CHOP ]
do

    let ROTATE=HEAD_CHOP-COUNT
    let IDENT=COUNT+1
    HOURMIN=`tail -$ROTATE $DATA_HOME/Datafile1 | head -1|cut -c 1-2,4-
5`
    KEY="k"$YEAR_JUL"."$HOURMIN$IDENT

    if [[ "$PASSED" != "-q" ]]
    then
    tail -$ROTATE $DATA_HOME/Datafile1 | head -1| sed "s/^/
$KEY\,$YES_DATE\,/g">>$DATA_HOME/Datafile2
    else
    queue_set
    fi

    let COUNT=COUNT+1

done
if [[ "$PASSED" != "-v" ]]
then
    AVERAGE=`cat ${FORMLOG}|grep Average|sed "s/[ ]*[ ]/,/g"|cut -f2- -
d,`
    KEY="k"$YEAR_JUL
    print $KEY","$YES_DATE","$AVERAGE|sed "s/\\\//g" >$DATA_HOME/
Datafile4
fi

sed "s/[ ]*[ ]/,/g" $DATA_HOME/Datafile2 > $DATA_HOME/Datafile3
#end
```

The scripts below comprise all the building blocks necessary for creating delimited text files.

1 **calcdate**

2 **control_load**

3 **format**.

You can produce all the necessary formatted text files by using the following command from the directory where the script files where placed:

```
control_load 1 -n -zz
```

This results in 21 files being placed in the directory defined as the location of your daily collection.

This article continues in next month's issue of *AIX Update*.

Robert Russell (UK) © Xephon 1998

AIX performance in client/server systems

This article outlines the building blocks of client/server systems, looking at the role of AIX in such systems and examining how the components affect performance. It also outlines some of the commercial tools for AIX that can be used to monitor client/server systems.

OVERVIEW

Many companies rely on AIX servers to provide business-critical functions. Increasingly, technical solutions are being built that use multiple client/server components. These are connected via LANs and WANs, and may also use middleware. The number of hardware and software components involved for processing even simple requests is growing almost exponentially. Although this provides considerable flexibility for delivering innovative and timely business solutions, it presents many issues for maintaining performance and service levels.

Nowadays the complexity of client/server architectures may make it extremely difficult to identify the exact bottleneck or bottlenecks that cause performance problems. Technical support staff typically have an ever increasing workload and may not have the time to identify the cause of a slowdown. However, for business-critical systems, it’s essential that problems are quickly identified and resolved.

CLIENT/SERVER ARCHITECTURES

Some years ago, the Gartner Group devised a method of classifying client/server systems into five categories. This method has since gained widespread acceptance. More recently new technologies have added to this picture. Distributed transaction processing, video on demand, and complex mixed-mode client/server systems at first seem difficult to classify using this system. Generally though, these are variations on the existing categories rather than completely new ones. The more common client/server systems can be classified as:

l *Distributed presentation*

This category comprises systems that exploit the video and graphics capabilities of intelligent clients (typically PCs) to improve the appearance of a traditional character-based Unix applications.

2 *Distributed data*

This category comprises client/server systems that provide unified and transparent access to data stored at various locations in the network. It includes two main types of application. The first is the *network server*. In this type of system, a workstation runs a software program, called a *redirector*, that intercepts attempts to access disks on network systems. It examines the drive name or letter and, where appropriate, redirects the disk access via the network to the server.

The second type of application is the distributed database, where subsets of the data reside on different database servers. When the user submits an SQL request to retrieve data, the request is examined to determine where the data is located. It may well span two or more servers. Each database is asked to return data in its subset and the results are merged and returned to the originator as one complete result set.

3 *Distributed logic*

There are many instances where end users collate information from several sources to build reports or interact with a number of systems in order to complete a single task. For instance, a stock system must be queried before an order can be placed in the ordering system. Many companies have written client applications that 'front-end' a number of server applications. These front-ends can be more user-friendly than server systems, they remove the need to toggle between server applications, and they can also add considerable extra intelligence that improves the speed at which an operator can interact with a customer.

TRANSACTION PROCESSING

Developers are creating ever more sophisticated client/server applications incorporating multiple servers and traditional host-based applications. Managing the integrity of these applications is becoming very complex. For instance, what happens if an application has opened several databases to write information and then a link is lost to one of them? This situation requires the application to cancel updates to all other databases. When this involves heterogeneous databases, possibly from more than one vendor, and a mixture of SQL-based and host-based applications, this back-out process can become very complex.

Frequently a transaction is a logical unit of work that may involve a number of applications on more than one system, with a transaction processing server managing all other applications. The server is responsible for synchronizing all applications, thus eliminating much of the effort involved with error handling. IBM's CICS is probably the best known transaction processing system, though others are available, such as Tuxedo and Encina.

NETWORKS

The network is the infrastructure in client/server systems. It is the glue that links clients and servers. As networks are required to handle more and more information, they are increasingly the bottleneck in client/server systems.

LAN topology: Ethernet, Token Ring, wireless

There are two predominant LAN topologies. Ethernet is the most common as a result of its low cost. Ethernet is fine for low to medium volumes, though it begins to struggle when network traffic approaches 45% of capacity. Most Ethernets run at 10 Mbps, though new technologies, such as 100BASE-T, provide 100 Mbps capacity. As a result of its design, Ethernet has an exponential degradation in performance with increasing volumes. This means that a level of traffic is reached when performance slows down severely.

For AIX systems the other widespread LAN topology is IBM's Token Ring. This provides between 4 Mbps and 16 Mbps of capacity. Token Ring's throughput tends to degrade linearly with increasing throughput, making performance problems more predictable than with Ethernet.

In order to extend the capacity of LANs in terms of both data and attached devices, break LANs down into smaller units and use repeaters, bridges, and routers to interconnect them. Note that although spare PCs make good bridges and routers, they often become bottlenecks.

WAN topology

Where physically remote LANs need to be interconnected, there are a number of options that tend to be grouped under the term WAN. The links can be based on telephone lines, ISDN, microwaves, radio, and fibre-optic cable. Typically cost dictates which type of link is chosen. These links are often far slower than LANs, so care must be taken to restrict the volume of data transmitted across them. ATM is an example of the new generation of high-speed connections for both WAN and links. ATM runs at 155 Mbps.

Protocol stack: TCP/IP, IPX/SPX, NetBIOS

This is the software running in both client and server that provides the ‘electronic envelopes’ used to exchange information. There are many different standards available, with TCP/IP, IPX/SPX, and NetBIOS being the predominant ones for client/server systems. Each of them has it strengths and weakness. For running across WANs, TCP/IP is the preferred standard. IPX/SPX often performs poorly across WANs, but is very effective on LANs. Performance is dependent on the interconnection between the LANs and WANs. Some routers possess a high degree of intelligence, and this can improve the performance of some protocols across WAN links significantly.

Many protocols, such as IBM’s SNA, regularly transmit a ‘polling message’. This is used to ensure that the host and the target are still in contact and that the link is not broken. When this message is received, an immediate response must be sent from the target system. If this response is not received within a given time period, the connection times out and the link is discontinued. This causes problems on some WAN that use public links, as they often charge per character and packet sent over the link. Such networks are typically optimized for large packets of data and don’t work as effectively when small packets containing just polling information are sent.

Networks based on X.25 are normally configured to send 128-byte packets. In order to send polling information, the network must be reconfigured to send smaller packets. This often limits network performance and increases charges. Some routers support a facility known as ‘spoofing’, whereby the router intercepts polling messages and emulates the response, thus keeping the link open without clogging up the network with low-value polling information.

Remote access: dial-up V34, ISDN

Remote users need to be able to dial in directly to servers or to attach to a gateway that emulates local network attachment. This allows them to access the facilities of a remote LAN. Remote connections tend to be fairly slow, typically 14.4 Kbps or 28.8 Kbps. This works well for e-mail and similar applications, though it’s too slow for data-intensive applications. ISDN is slowly emerging as an alternative that offers greater speed, though it suffers from high initial connection costs that have limited its acceptance. In the UK, ISDN provides two 64 Kbps circuits plus a low speed voice and signalling circuit. 56 Kbps modems are now beginning to make inroads into the market and 64 Kbps modems are on the horizon. Such developments are likely to limit ISDN’s role.

MIDDLEWARE

Middleware is a transparent layer of software that isolates applications from the complexities of the underlying hardware and networks. This new-ish term covers a large range of technologies. Note that some database vendors have hijacked this term, using it to refer to software that allows various clients and ‘foreign’ databases to connect into their database. Middleware is a much wider subject than this, being a strategic component in the building of complex client/server systems. It’s difficult to provide an exact outline of all of the areas covered by middleware, but the following list gives a general overview:

- *Data access*. This is the traditional type of middleware used to enable a wide range of clients to access SQL databases. It can also be used to provide local access to files held on remote systems. It provides a generic interface for applications to access data easily, regardless of the data's location. Data access drivers sit on top of the networking software. Examples include Microsoft's Open DataBase Connectivity (ODBC) and the Network File System (NFS).

- *Object brokers*. These are used to enable applications to be built where the actual application code resides on different servers. When an application calls a particular procedure or subroutine, this may not be on the local server. The object broker intercepts the call and routes it to the system where the code is resident. The code is then executed and the results returned to the local system. In concept this is similar to the Dynamic Link Libraries (DLLs) used in many multitasking systems such as OS/2, Windows NT, and Unix. Typically object brokers are used to link different implementations of C++, where inconsistencies in link editing, for example, normally cause problems. CORBA is a standard for object brokers from various vendors.

- *Distributed transaction processing*. Although transaction processing is really a form of client/server, it does have a role to play in linking computer systems together. It can be used as the glue to link and manage complex client/server transactions in mission critical environments.

- *Messaging (MOM)*. This lets applications communicate using simple structured messages. In many ways it's similar to e-mail, to the extent that some messaging middleware uses e-mail to transport its messages. It is by its nature asynchronous, thus providing applications with a means of handling delays when waiting for responses from other applications.

- *E-mail*. This is an effective means of exchanging information between applications. E-mail facilities are a standard part of most organizations' IT infrastructure. There are a range of programming interfaces that enable developers to write co-operative applications. E-mail provides a low cost way of linking applications where e-mail delivery speeds are acceptable. MAPI/VIM are standard interfaces for applications to communicate via e-mail.

- *Screen scrapers*. These middleware programs are used to interface with character-based Unix applications. The program can be used to present the information in a more graphical and user friendly format and it can also be used to automate the exchange of information between incompatible systems.
- *Encapsulation*. This is based on screen scraping, but uses SQL-type statements to enable developers to create applications that access data in character based systems. An application server is typically used that looks like a traditional database server. This interfaces with Unix systems. When SQL requests are received, it captures screens from these systems and returns them as a result set back to the requester.
- *Transparent resource access*. DCE is an example of a product that spans a range of middleware categories. It is designed to isolate the user from the source of the information with which they work. It provides features such as NetWare-like file access, Kerberos security, distributed printing, and date and time services.
- *File access*. This works by intercepting local file access requests and re-routing them to a remote server. This process is often referred to as redirection. It enables workstations to access a considerably wider range of information than is available locally. Typical enabling products include Novell NetWare and Microsoft LAN Server. Under AIX, NFS is a good example of a file access product.

This is only a partial list of middleware types (it is reckoned there are eighteen types in all). However, a full discussion of the more obscure products is outside of the scope of this article.

FILE SERVERS

Fileservers such as those based on NFS, Novell NetWare, and Microsoft LAN Server have become the predominant means for sharing information on LANs. While they are still complete computer systems in their own right, and their performance is still dependent on the balance between the CPU, memory, I/O, as the workload resulting from requests they receive via the network, you also need to consider the following factors when investigating performance bottlenecks:

- *LAN adapters*. There are a wide range of LAN adapters in the marketplace for Token Ring, Ethernet, and emerging standards such as 100BASE-T. There are considerable performance variations between different adapters.
- *Buffers*. Network cards have a small amount of memory that is used to hold microcode or buffer information being transferred to and from the network. In some systems, it is possible to alter the memory allocated to each service being using via the network adapter. Most Unix systems do not allow this to be altered, but PC systems generally do. This allocation of memory can impact performance.
- *Cache*. Accessing information on disk drives is slower than accessing information in memory by a few orders of magnitude. File servers attempt to pre-fetch information from the disk drives and hold it in memory. For instance, when a sequential file is opened, the file server attempts to read the entire file into memory. When the application asks for the next record in the file, this is sent via memory rather than from disk. This is caching. For many servers, the amount of memory allocated to the cache has a considerable impact on performance. Servers provide statistics that detail the ratio of read requests met from cache against the percentage that required physical I/O. Where a low percentage is being met from cache, increasing the memory can provide worthwhile performance improvements. It is also possible to define the amount of memory allocated to caching on some Unix systems. If there's spare capacity, then it may be worth allocating more memory to caching.
- *I/O subsystem*. This is the connection between the disk drives, LAN adapters, and the processor. The speed of this link can be a bottleneck. There are two main components, one being the connection between the peripheral device and the computer's data bus. Modern high-performance disk drives tend to use the SCSI standard as the means of accessing the drives, though for AIX SSA is a higher-performance alternative. There are other proprietary formats, but they are fairly rare compared with SCSI.

PC servers tend to use a mixture of IDE, EIDE, and SCSI. IDE has largely been superseded by the enhanced IDE standard, or EIDE, which provides mid-range SCSI-type performance at a lower cost. SCSI is the preferred choice for higher transfer rates and for large disk drives.

- *Data path to user (hops and routers)*. Even if the server is suitably configured, its potential may never be realized if it is constrained by the network linking the client to the server. As networks grow in size and complexity, it is rare for users to be connected directly to the same LAN as the server. The intermediate links can often constrain throughput. Modern routers are capable of very high throughput, but tend to be expensive. Frequently servers are also used as routers and old, low specification PCs are commonly used as bridges for no other reason than because they are available. Though cost-effective, Intel-286 or 386-based PCs are often under specified for the job and cause severe bottlenecks; their replacement with faster PCs or dedicated routers and bridges can bring significant performance gains. A quick survey of the network, paying particular attention to these intermediate links, can soon identify bottlenecks. A LAN analysis tool is particularly useful for collecting evidence as to where problems lie. Some modern network management tools, such as IBM's NetView 6000, can provide useful information as well.

DATABASE SERVERS

The database server is a blend of file server and application server, dedicated to retrieving relational information via SQL requests. Tuning these servers is a complex task that has been described in a number of good books. This section is therefore very much of an overview. Every database requires different methods to resolve performance issues, though they all have common features. Overall performance is governed by the balance between the CPU, memory, and I/O in relation to the workload, as is the case with file servers. Performance issues can be split into data design and implementation. Some of the common ones are:

- *Data design*. The way in which the data is organized is the single most important factor in determining performance. Careful consideration needs to be given as to how information is to be retrieved and an efficient strategy needs to be devised to meet these needs. Formal data analysis and normalization are very important. Although these are often regarded as ‘chores’, they have a critical bearing on performance.

- *Symmetrical Multi-Processing (SMP)*. IBM’s SP/2 servers support multiple-processor configurations – a feature that’s also supported by AIX. Many databases, such as Oracle, can take advantage of SMP to allow processor power to be expanded easily. SMP also allows load balancing, allocating tasks evenly across the processors. Additionally, with many databases you’ll find that most of the workload is being carried out by only a few processes. SMP allows each of these processes to be allocated its own processor, providing considerable improvements in performance.
- *Caching*. Databases make heavy demands on I/O systems when reading and writing large quantities of data. Caching can make a considerable difference to this. Always configure the cache in line with the manufacturer’s recommendations. The ratio of cache hits to physical reads is very important and should be monitored carefully. Memory is often the single largest physical factor in determining performance. Prices for memory are decreasing rapidly, so it’s now usually more cost-effective to improve performance by adding RAM than by any other single means.
- *Indexes*. Records in tables are accessed using keys. You can either find the record by searching the data stored on the disk drive itself or by searching a replica list of keys kept in memory. The memory-based key is referred to as an index. Searching an index is many times faster than using a disk-based key. As indices consume memory, their number is normally kept to a minimum. However, where a table is regularly accessed through a particular key, it is worth making this an index. The difference in access times when using an indexed key and a physical key can be an order of magnitude or more. Ideally you should study data access patterns to determine which keys should be indexed.

- *SQL*. In this context, SQL can be regarded as the language used to specify the data that needs to be retrieved or inserted into the database. As with most programming languages, there are usually many ways of coding the same query, and the exact form of the statement often has an impact on performance. There is often a considerable difference in the SQL written by experienced and inexperienced staff. Particular care should be taken to make sure that indexed keys are used wherever possible.

- *Optimization*. Modern databases provide facilities to translate an SQL query into the optimum method to retrieve the required information. This is often referred to as *query optimization*. The database follows a set of rules known as a *strategy* to determine how to retrieve data. The optimizer can improve performance, though it tends to struggle when working on poorly written queries as it has only a limited 'knowledge' of the data. Most databases, such as Sybase produce a log showing the optimization strategy selected for each query. This can be a useful reference when trying to tune queries as it often highlights areas where indexes and other techniques may help.

- *Distributed data*. Modern databases allow the designer to split the data itself between a number of servers. This is useful when information is derived from a number of locations, but needs to be centrally accessible. Although distributed databases usually offer 'location transparency', this usually has an impact on performance. A query performed on a local server is much faster than one on a remote server. When queries require data to be retrieved from several different systems, the speed of the query is dependent on the slowest link in the chain.

- *Replication*. Problems with accessing distributed databases have led to the development of an alternative strategy known as *replication*. This involves keeping copies of the data at remote sites. Replication works best when read-only access to data is required at the replication sites, as it provides rapid access to the data. If this is not the case, then replication makes it necessary to synchronize the various copies of the data, which can be a complex issue if the information can be updated on both the replicated and the master databases. Synchronization requires the use of 'two-phase commits' or other similar strategies. A workaround is to allow only read-only access to replicated copies, allowing updates only on the master.

- *LAN adapter*. As is the case with file servers, the throughput of the LAN adapter can be a bottleneck. The **netstat** command may be used to provide useful information about the LAN adapter's performance. If necessary, it is usually possible to add a second adapter to increase the throughput.

- *Mirroring*. With business critical systems, mirroring is often used to provide protection against disk failure. Mirroring simply copies data between the main disk drive and a drive used as a back-up. Mirroring can be performed by either hardware or software, depending on the system and application. The trend is towards using RAID-based storage, where an intelligent SCSI disk drive controller automatically copies the data between different drives. If one fails, data is still available. Software replication performs the same function, but incurs a CPU overhead. Where possible use hardware caching as this improves performance.

- *Locking strategy with multi server/domain queries*. Although databases may provide concurrent access to the same data for many users, they only allow one user to update the information at any time. When a table is opened for updating, the database typically locks the record concerned to prevent two users from simultaneously updating it. Depending on the database, a number of different locking strategies can be used to ensure data integrity. This includes column, record, and table locking. The chosen method is usually a compromise between ensuring data integrity and minimizing the impact on other users who wish to update the same information. Once a transaction has locked a record, another user wishing to update the record must wait until the lock is released. With interactive transactions the delay can be considerable. Another possibility is that two applications need to access the same set of records in different tables. Then, it's possible to get a deadlock where each application tries to access the same files.

When a query is on a distributed database, all components of the database must be locked until all updates are complete. This can have a considerable impact on performance when some systems are connected by slow links, so this needs to be considered at design time.

This article concludes in next month's issue of *AIX Update*.

Steve Butler (UK) © Xephon 1998

Contributing to *AIX Update*

AIX Update is primarily written by practising AIX specialists in user organizations – not journalists, or consultants, or marketing people. In our view, the information and advice provided by such people – people like you and your colleagues – are far more valuable to their fellow professionals than the alternatives available from other sources.

We don't expect you to write an original article from scratch – just send us listings or specifications of any relevant programs, utilities, scripts, user modifications, or other code that might be of use to other installations, with a short explanation of why it was developed and what it does. And most IS departments produce a great many internal technical reports and other documents, many of which can easily be adapted for publication.

Contributors aren't just helping their fellow professionals – they also receive a significant material reward themselves. We pay good rates for the articles we publish: $250 (£170) per 1000 words if we get copyright, and $140 (£90) per 100 lines of code.

A copy of *Notes for contributors* can be obtained from Xephon's Web site at *www.xephon.com.*

AIX news

IBM has announced Version 3.6 of its C and C++ compilers for AIX, OS/2, and Windows NT. The new tools aim to provide cross-platform portability – the AIX version supports applications developed using IBM C Set++ for AIX Version 3.1.4, and once applications are migrated to the new compiler, they can be ported to either OS/2 or NT.

Version 3.6 also supports the creation, execution, and debugging of 64-bit C applications for AIX 4.3. Prices start at US$1,899.

IBM has also begun shipping Torrent Systems' Orchestrate, a tool for building and deploying parallel programs. The software supports SMP and SP systems running AIX, and allows systems to be written that are fully capable of running in parallel against large volumes of record-oriented transaction data. Prices start at $7,800 per developer.

For further details, contact your local IBM representative.

* * *

Compuware has announced that Uniface applications now run on AIX Version 4.3 with full support for new OS features, such as interoperability between 32-bit and 64-bit applications.

For further information contact:
Compuware Corp, 31440 Northwestern Highway, PO Box 9080, Farmington Hills, MI 48334, USA
Tel: +1 810 737 7300
Fax: +1 810 737 7199
Web: www.compuware.com

Compuware Ltd, 163 Bath Road, Slough SL1 4AA, UK
Tel: +44 1753 774000
Fax: +44 1753 774200

* * *

SAS Institute has launched a universal ODBC driver designed to make SAS data available to ODBC-compliant applications, making it possible to access such data without SAS software. U-ODBC lets users open and query SAS data sources resident on AIX from desktop platforms. ODBC drivers are also available for other Unix platforms, OS/2, Macintosh, and DEC OpenVMS, VAX, and Alpha.

For further information contact:
SAS Institute, SAS Campus Dr, Cary, NC 27513, USA
Tel: +1 919 677 8000
Fax: +1 919 677 8123
Web: www.sas.com

SAS Institute, Wittington House, Henley Road, Medmenham, Marlow, Bucks SL7 2EB
Tel: +44 1628 486933
Fax: +44 1628 483203

xephon