34

AIX update

August 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

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).

I/O pacing

I/O pacing is best described as a black art. While some system administrators hide at the mere mention of its name, others regard it as the best thing since sliced bread.

I/O pacing is used to solve (or, occasionally, make worse) a specific type of problem. Processes that are very I/O intensive and generate a large amount of disk I/O can seriously affect the response time of less I/O-intensive processes. Large I/O jobs can create megabyte-sized FIFO queues, to the end of which jobs with modest I/O requirements must add their requests. This is the problem that I/O pacing tackles – the blocking of I/O by a few rogue processes. I/O pacing attempts to regulate the amount of disk I/O that a single process can 'push' onto the I/O queue.

Note the two qualifying terms above: 'or make worse' and 'attempts to regulate'. I/O pacing isn't always the best solution. While it can easily make performance worse for I/O-intensive processes, it can also degrade the performance of less I/O-intensive jobs.

I/O pacing works by enforcing high and low 'water marks'. A process can 'pump' I/O until the amount of pending I/O reaches the high water mark. The process must then wait for pending I/O to be de-staged to disk. Once the amount of pending I/O reaches the low water mark, the process can pump more I/O. The benefit of this approach is that other processes are able to send and receive I/O while the offending process sleeps.

The simplest thing about I/O pacing is the actual mechanics of setting high and low water marks – from **smit** choose *System Environments* then *Change/Show Characteristics of Operating System*. There you'll find *High Water Mark* and *Low Water Mark* fields. Note that setting both the high and low water marks to 0 effectively disables I/O pacing.

Choosing the best values for high and low water marks is where the problem lies. There are no hard-and-fast rules that govern these settings. Experimentation is the only method by which they can be established. It's also worth bearing in mind that the optimum values of water marks on one system are almost inevitably different (often very different) from the optimum values on another, making it difficult to configure systems using the same or similar settings. Why are these settings so different on different systems? It's simply a matter of I/O patterns – different boxes are laid out differently, they have different throughputs, different processes running at different times, and differ in numerous other ways that affect I/O. In fact, the I/O pattern on your system today may be completely different from that in six months' time. So, to refine my comment above, continuous experimentation is the only method of establishing water marks.

To show the effect of I/O pacing, I set up a simple test. Given that a large **cp** is very I/O intensive, I copy a large file from one disk to another. While **cp** is running, **ls** is run on the **cp** destination directory. **ls** is run in a continuous loop, with a five-second gap between iterations, and is timed. By examining the time required to run **ls** we can see the effect that the large **cp** has on it. The test program is:

```
/usr/bin/timex 2> cptime cp /test/bigfile /disk2/test &
while true
do
/usr/bin/timex 2> dummy ls -l /disk2/test
cat dummy >> lstime
sleep 5
done
```

Note that the above program also times the **cp** command. While I/O pacing may improve the time it takes to execute a process that incurs only a light I/O load (**ls** in this case), in so doing it may have an adverse effect on the execution of a process that incurs a heavy I/O load (**cp** in this case). To measure the effect of I/O pacing, I run the test program and pick out the ten worst timings of the **ls** command. Getting the average of the ten worst timings gives a good indication of how the heavy I/O load affects **ls**. Running the test program with different high and low water marks shows the effect of I/O pacing on both processes.

THE RESULTS

		(Average of ten worst results)	
High Water	Low Water	ls Time	cp Time
0	0	0.609	165.44

High Water	Low Water	ls Time	cp Time
20	10	2.023	293.17
40	10	4.022	301.40
60	10	0.063	199.36
60	30	0.130	192.48
60	50	0.148	179.20

When examining the above figures, bear in mind that the response times are averages, which has the result of masking some very poor results – in one case (using a high/low water mark of 40/10), **ls**'s worst response time was nearly 19 seconds. The best way to examine these results is by means of charts – see Figures 1 to 4.

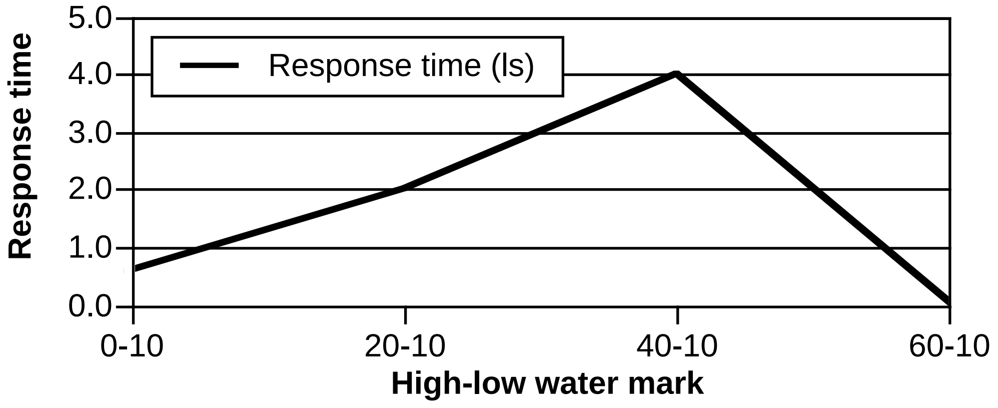

Figure 1: Effect of increasing high water mark on **ls**

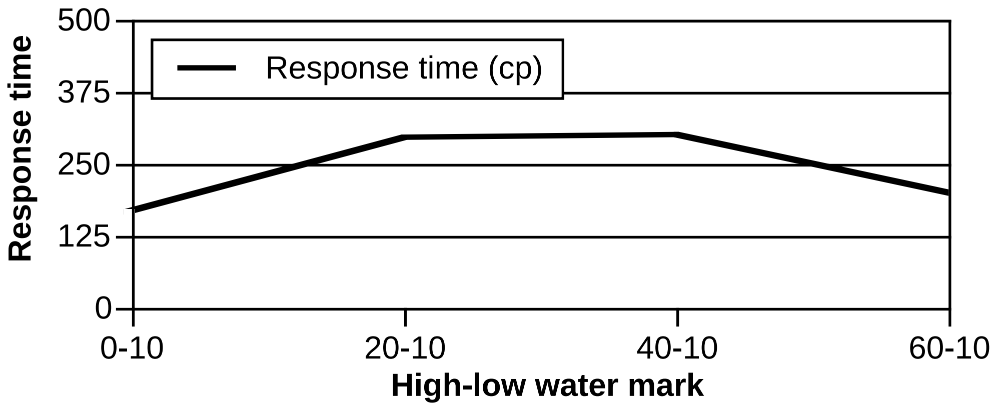

Figure 2: Effect of increasing high water mark on **cp**

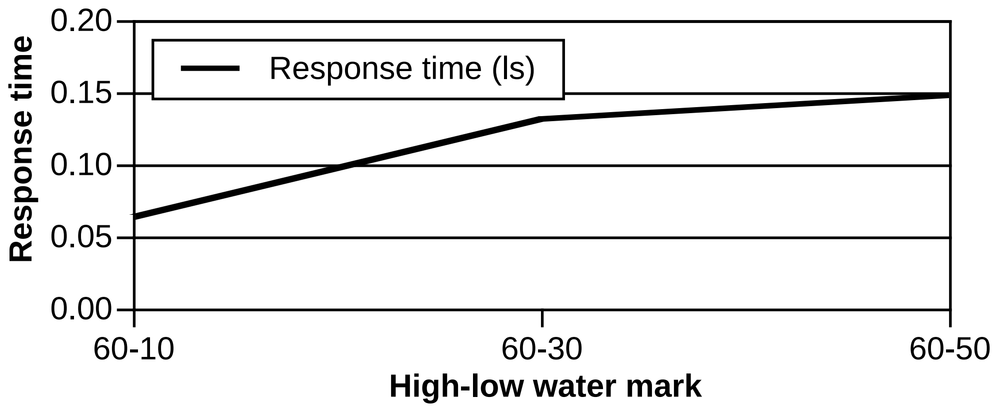

Figure 3: Effect of increasing low water mark on **ls**

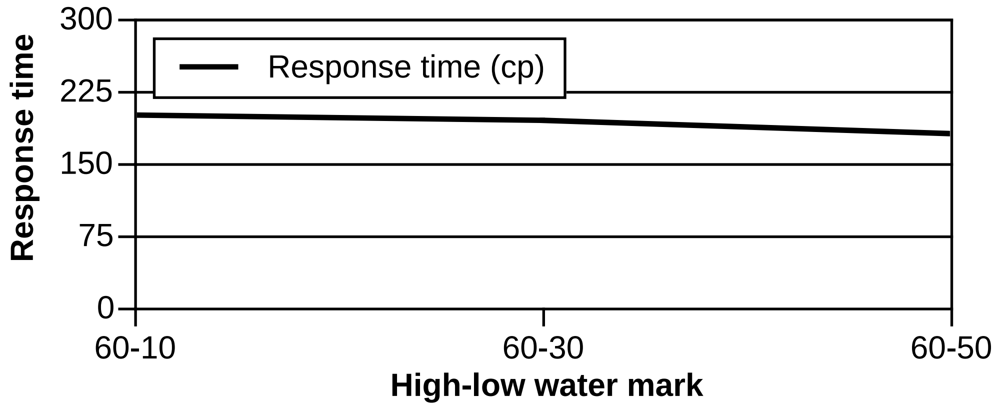

Figure 4: Effect of increasing low water mark on **cp**

INTERPRETING THE RESULTS

The optimum setting in the above example is a high water mark of 60 and a low water mark of 10. Using these settings, the response time of both the **cp** and **ls** commands are near their lowest values.

Notice that, as the high water mark increases, so does **ls**’s response time. The reason for this is that **cp** is now allowed more pending I/O, which blocks **ls**’s I/O. However, this changes dramatically at high/low water mark values of 60/10, at which point **ls**’s response time decreases dramatically. I/O from **cp** is allowed to build up for a long period (thus **cp** is faster), but must wait longer to de-stage to the low water mark, which allows more time for **ls**'s I/O. In effect both processes run faster.

Increasing the low water mark has a negative effect on **ls** (its response time increases) while having a positive effect on **cp** (lower response time). As the low water mark increases, the **cp** process has to sleep less – it doesn't have to wait as long for pending I/O to reach the low water mark.

In my opinion, 60/10 is the optimum value in this example. However, there are others who would claim that the optimum value is 0/0, as this allows **cp** to run fastest – as long as you don't care about the time it takes to run other processes' **ls**. Others would consider that 60/10 is the best value as this is the value that allows **ls** to run the fastest, regardless of whether it also works for **cp**.

What you find to be the optimum value depends on the type of I/O you want to favour. Is the performance of I/O-intensive processes more important than that of small jobs, or is the opposite the case? Do you want to achieve a balance between heavy and light I/O processes? Each system administrator must make this decision when determining how to tune I/O at their installation.

John McAvoy (Ireland) © Xephon 1998

Securing and optimizing AIX's root user

INTRODUCTION

One of the most common problems with Unix installations, including ones running AIX, is the fact that the almighty *root* user cannot be restricted in the same way as is possible with some other operating systems. For instance, MVS administrators can use RACF to define a 'special user' that has only selected privileges. However, there are certain precautions that every administrator can take in order to secure the *root* account from accidental or malicious misuse.

CHARACTERISTICS OF THE ROOT USER

The *root* user (or, to be precise, the user that has userid zero) is an architectural limitation to the security of every implementation of Unix, including AIX. The existence of *root* makes it difficult to implement Unix with the same level of security as we've become accustomed to on mainframes and a number of other platforms. On the one hand, everybody knows that the *root* account should be used exclusively for special tasks that require permissions available only to the *root* user. On the other hand, when you work with Unix on a day-to-day basis, you get accustomed to the fact that an administrator needs to work with *root* almost all of the time. In addition, it's not just Unix administration that requires permissions available only to *root* – some applications and infrastructure services, such as a name server or Web server, often can be run and managed only by *root*.

USING PERSONAL ACCOUNTS

An easy first step is to convince users to use only personal accounts (like 'mabel', for 'Michael Abel') to logon, instead of logging on using common IDs, such as *root*. As users become accustomed to logging on using their own accounts, it becomes easier to restrict the use of the *root* account. If a user really needs to work with *root* privileges, he or she can use the **su** ('switch user') command. A simple way to force people to use personal accounts is to restrict direct access to *root*.

We recommend that you change and verify the following attribute of *root* in order to force people to login with their personal IDs:

```
chuser -a login=true root
```

This is a precaution to ensure that the default setting that allows *root* to logon at the system console has not been changed. Once this is set you can switch off all other ways to logon as *root*. This measure is most effective in organizations where systems are located in a special area to which access is controlled either by key or a badge reader.

```
chuser -a rlogin=false root
```

This makes it impossible to logon as *root* using **rlogin** or **telnet** commands over the network. It also disables logons over the LAN using CDE or XDM, ensuring that you have to logon using your personal account instead.

```
chuser -a su=true
```

This ensures that users can use **su** to assume the identity of the *root* user. In other words, users logon using their personal accounts and then use the command **su *-password*** (where ***password*** is root's password) to become root.

```
chuser -a sugroups=aixperts
```

This last command ensures that only members of a designated group ('aixperts' in this example) can use the **su** command to access the root account (which also requires them to know *root*'s password). It's possible to list more than one group in the above command (for instance 'aixperts, senioradms, operators') in order to provide the privilege to members of the listed groups.

Changing these four *root* account settings implements our policy of restricting direct access to the *root* user. Once these measures are in place, only members of a special group are allowed to access *root* by first logging in with their personal ID and password and then switching to *root*. During the **su** operation the user has to type in the *root* password, which means that access to the system is secured firstly by the user's own password and then by the root account's password.

FURTHER RESTRICTING ROOT ACCESS

Systems that require even stricter access control to the *root* account may still use the procedure detailed above if they modify it in the following way, which adds a few more restrictions to the operations that may be carried out with **su** to get *root* permissions.

As indicated above, access to the *root* account is restricted to users that are able to issue the **su** command, which means that one has to be a member of a special group to access *root*. In addition the user also has to know the *root* password, as this has to be typed in during the **su** operation.

Now, to tighten control even more we create a special user who's responsible for security administration and is able to use only three commands (the commands are implemented as shell scripts using the **sudo** command, as described below):

```
lsadmin

mkadmin

rmadmin
```

These commands are used to display or change settings that control whether a user is permitted to use the **su** command to access the *root* account. In our example **lsadmin** lists all members of the group 'aixperts', **mkadmin** adds a user to this group, and **rmadmin** removes a user from this group. Using this set of commands allows the security administrator to control which users are allowed *root* permissions. The scripts may be implemented as front-ends to AIX commands that manage users and groups (or, to be precise, the membership of groups).

In reality this means that even an administrator who knows the *root* password can't always access the *root* account on a mission-critical system – the administrator first has to obtain this authority from the security administrator, who is able to collect information about systems on which *root* permissions have being granted and the purpose for which they are granted. Once this information is documented, he can then open a *root* account for the administrator.

Using this technique allows us to distinguish between production systems, where it is desirable to reduce changes to a minimum, and 'non-critical' systems, such as those undergoing installation and customization, on which *root* is used heavily and doesn't need to be controlled by security administration.

USING SUDO (SUPERUSER DO)

Unix, including AIX, provides the **suid** ('set userid') command that allows programs to be executed with *root* privileges even when they are started by a user logged on using a normal account. An example of this is the **passwd** binary, which is used to change the user passwords and has a suid bit set, as demonstrated by the following exchange:

```
which passwd

/usr/bin/passwd

ls -l /usr/bin/passwd

-r-sr-xr-x   1  root   security   11846 Feb 06 1998   /usr/bin/passwd
```

So, when a regular user such as *mabel* uses the **passwd** command, the command runs under the effective userid of the owner of the program (*root*). This is necessary because the file that contains *mabel*'s password (*/etc/security/passwd* on AIX systems) may be written to only by *root*. The **suid** trick is also used for printing and some administrative commands – in each instance, the user that starts the command gets more permissions because the effective userid is switched to the owner of the program that's executing.

sudo, a freeware utility, extends the scope of the 'suid trick' by allowing programs that don't have the suid bit set to run with *root* privileges. Instead of directly calling the command the user wants to execute, the user issues the **sudo** command followed by the actual command. During execution, **sudo** first checks the file */etc/sudoers* to establish whether the user is allowed to execute the program. Next **sudo** creates a process that runs with *root* authority, which it uses to 'exec' the command the user actually wants to run.

Let's implement a **sudo**-based solution to the problem of allowing users that don't have access to the *root* password to shut down the system. In the control file */etc/sudoers,* which is created when **sudo** is installed, we add the following lines:

```
# sudoers file.
#
# This file MUST be edited with the 'visudo' command as root.
#
# See the man page for details on how to write a sudoers file.
#

# Host alias specification
Host_Alias      MYHOSTS=host1, host2, host3

# User alias specification
User_Alias      MYGROUP=mabel

# Cmnd alias specification
Cmnd_Alias      MYCMDS=/etc/shutdown
```

```
# User privilege specification
root            ALL=ALL
MYGROUP         MYHOSTS=MYCMDS
```

If the user *mabel* tries to issue the **shutdown** command below, he or she will receive an error message.

```
     shutdown -F
```

As this program has no execute permission for *mabel*, the shell won't start it. However, using **sudo** to start the program, *mabel* is able to run the program with *root* permissions and shut down the system:

```
     sudo /etc/shutdown

     We trust you have received the usual lecture from the local
     System Administrator. It usually boils down to these two things:

             #1) Respect the privacy of others.

             #2) Think before you type.

     Password:
```

As you can see in the above example, **sudo** issues a warning message and requires the user's password (though not *root*'s password) to be typed in.

I find that **sudo** is a good tool for systems where only a small number of commands and scripts have to be run with *root* permission by a group of people who have only normal accounts. Groups like operators, security administrators (see above), and users who operate printers, work as part-time administrator, etc are good examples of those whose need to run some programs with *root* permission may be handled with **sudo**.

You may be interested to know that **sudo** is distributed as source code, which means that it may be easily tailored in many ways to suit your installation. For a quick solution point your browser to *http://www.bull.de/pub/* (note the trailing forward slash), which is an excellent source of pre-compiled and pre-configured shareware tools for AIX 4.*x*. All tools are packed in **installp** format – just download, decompress, and use **smit** to install them. Another freeware archive is available at *ftp://aixpdslib.seas.ucla.edu*. Further tips and tricks are also available on our Web site at *http://www.resnova.de*.

USING SPECIAL GROUPS

Another strategy for avoiding unnecessary use of the *root* account is to use the pre-defined groups that are available in AIX. As a member of these groups, a non-root user is capable of actions normally limited to *root*. Let's take printing as an example. Every user is allowed to use the command **qprt** to start a print job. Every user is also able to use the **qchk** command to obtain the job number of their print job(s), so they can use the command below to cancel them, if required.

```
qcan -x JOBNUMBER
```

But what if you'd like to create a 'printer administrator' who's able to cancel print jobs belonging to other users and is also able to start and stop print queues? The two solutions to this problem are to let the print administrator use the *root* account and to make your print administrator a member of the *printq* group.

Being a member of *printq* enables normal users to act as system administrators with respect to all tasks that involve printers, print queues, and print jobs. This is a good solution to the problem of providing designated users with enough authority to allow them to carry out day-to-day print operations without having to provide them with the *root* password.

Other examples of special groups available in AIX are *security* (which covers some aspects of user administration) and *system* (which allows designated users to mount CD-ROMs). One drawback should be mentioned: these groups grant their members access to all objects of the same type, so that, for instance, print operators have access to all printers – there is no way to restrict permission to a subset of printers. Thus a printer administrator who's a member of *printq* is able to cancel any print job on any queue in the entire system. The ability to have a printer administrator for each printer is not yet implemented on AIX, though some third-party print management software products have implemented this facility.

ADMINISTRATIVE ROLES

To quote from the appropriate AIX manual: "roles consist of authorizations that allow a user to execute functions normally requiring root user permission”. This is what we’re talking about, isn’t it? Starting with AIX 4.2.1, IBM now provides some pre-defined roles that may be used to delegate *root* user permissions to other users. Here is a list of the roles that are available:

- Manage all or basis users
- Manage all or basis users passwords
- Manage roles
- Manage backup and restore
- Manage shutdown.

A user may assume one or more of these roles in order to perform functions related to the role. Roles allow administrators to delegate authority to normal users to allow them to carry out a number of administrative roles, as discussed above, without using either special groups or **sudo**.

There are currently two limitations of roles of which you should be aware: administrative roles are only available on AIX 4.2.1 (and higher) and administrative roles are implemented in AIX commands themselves, which means that there is no way to make other commands ‘role-enabled’ (by contrast, **sudo** works with every binary command and shell script executable). The first restriction is (at the moment) one that has to be taken seriously – there are numerous installations around that are still using AIX 4.1 (or even AIX 3), even though they may have already rolled out a number of AIX 4.2 or 4.3 systems. We recommend you implement a solution that is applicable to all AIX systems used in the installation, otherwise you’ll encounter compatibility problems later on.

TIPS AND TRICKS

A filesystem for root

When you install AIX, the home directory of the *root* user is by default the system’s root directory. On many systems this can cause two possible problems.

The first problem is that using tools like **smit** when you're logged on as *root* results in many large log files in *root*'s home directory. Since AIX requires a certain amount of free space in the root filesystem (for instance, during reboot), it's a good idea to keep *root*'s private data separate from system areas. Implementing a separate home directory for *root* avoids this problem, which could result in the AIX system halting unexpectedly during a system boot as a result of the root filesystem being full.

The second problem is that no-one is as perfect as they would like to be. Even experienced administrators may accidentally issue the command **rm -fr ***. If the user is working in their private home directory, this results in the deletion only of private data. However, if the user is logged on as *root* and is working in the root directory, this error results in disaster.

To avoid these two problems I recommend that you set up a separate filesystem for *root*, for instance by creating */home/root*. This ensures that every problem caused by *root* is local to the new filesystem and does not result in problems either for other users or for the system itself. Below is a step-by-step guide of how to do this.

1 Set up a new filesystem and mount it:

```
crfs -v jfs -g rootvg -a size=15000 -m /home/root -A yes

mount /home/root
```

2 Copy existing 'root-dot' files to the new filesystem:

```
cd /home/root

cp /.* .
```

3 Change *root*'s user attributes:

```
chuser home=/home/root root
```

Verify your changes by looking at */etc/passwd* and then try to login with the modified account. Check with *pwd* to ensure that */home/root* is your new working directory and verify that the environment is set up correctly (for instance, check the PATH variable to ensure that it's right).

Creating an emergency root user

If people start to avoid logging on as the *root* user as a result of measures you've taken, there is a danger that at some time access may be needed to the *root* account and there is no-one around who knows *root*'s password. In such an instance (and to cater for other emergencies in which the *root* account may not be available) I recommend that you have a second *root* account ready.

The problem is that neither **smit** nor the **mkuser** command can be used to create an account that has *root*'s userid (0). The easiest way around this is to edit the */etc/passwd* file, copy the line that contains the *root* definition, and paste it back into the file, so it contains two copies of the line. Change the name of the duplicate account and change its home directory to '/' (in case the */home/root* filesystem is not available).

Here is an example of what */etc/passwd* should look like after the necessary changes:

```
root:!:0:0:Superuser:/home/root:/bin/ksh

helpme:!:0:0:Superuser emergency account:/:/bin/ksh
```

I've called the duplicate *root* account *helpme*, but feel free to name it as you like. Many of our customers use an abbreviation of the company's name (for instance, *rncroot* for *res nova consult*) for the emergency *root* user.

After creating the account, we give it a password that's hard to remember and test the login. A copy of the password is then kept in a safe location (security guards can help here; also consider keeping the password where emergency keys are stored). If anyone needs immediate access to a system at any time and the normal *root* account is not accessible, one may rely on that back-up login.

Michael Abel
res nova Unternehmensberatung GmbH (Germany)

Date and time manipulation

This month's instalment concludes this article on date and time manipulation using **calcdate_diff** *and* **calcdate_time**. *The first part of this article appeared in last month's issue.*

EXAMPLES OF IMPLEMENTING THE SCRIPTS

Batch milestone checks

I'll begin by looking at one of the utilities designed to monitor overnight batch jobs. At our site, we have a schedule of overnight batch jobs that runs every night from Monday to Friday. The timing of this schedule is very important as any problems would have a major impact on our business. The batch work needs to be monitored as it progresses, and must reach certain 'milestones' before set times. We have a monitoring system that sends an alert message to the administrator on call if any milestones are reached later than expected. Monitoring these events is crucial to keeping batch jobs on schedule.

However, a number of designated users can vary the range, depth, and policy of overnight batch processing work. As a result, the exact timing and duration of batch jobs is neither constant nor under the direct control of administrators, and this places even more emphasis on the need to monitor the activity and timing of overnight batch work.

We noticed that problems arise during the year in the run-up to 'period-ends' (month-ends, year-ends, etc) when batch work takes longer to complete. When this happens, certain milestones in our overnight batch work fall behind schedule, which results in the pager alert system being invoked. This, in turn, results in the administrator on call being disturbed and signing onto the system only to find everything running fine but behind schedule. To allow for this we decided to keep an eye on overnight batch work and to make schedules flexible so they can adapt to increased execution time resulting from additional work.

The utilities below were set in place to produce a daily report of milestone events. This report quickly shows if any processes are beginning to take longer to complete. The main utility calculates the interval between milestone events using the **calcdate_diff** and **calcdate_time** scripts.

Our overnight batch work produces a set of nine files that are used for timing milestones (the files are used to calculate the elapsed time between successive stages of the batch process). The script uses the time stamp generated by the **ls -l** command to determine timings. Although our implementation also uses information from within the files themselves, it nevertheless provides a workable example of how to use **calcdate_diff** and **calcdate_time** for the purpose of this article.

The nine files are produced as a result of certain events during the batch run and must be on the system by the scheduled completion time (the files are automatically deleted before each run to ensure that the previous run's files aren't accidentally used). To give you an idea of what's in the files, the first one contains a time stamp from the beginning of the run and the last one contains the time from the end of the run. The files are called *build_start*, *build_1.fin*, *build_2.fin*, through to *build_8.fin* (the last file produced each night). The script listed below produces a delimited text file showing the number of minutes between the creation time of each of the files on the system, eg:

```
Key, Date, Build_1.fin, Build_2.fin....,Build_8.fin
k98001,01/01/1998,120,42....,112
k98002,02/01/1998,120,43....,112
etc
```

You may be interested in knowing that this file is in the same format as described in my article on monitoring system performance (*Monitoring AIX with PCs*, *AIX Update* Issues 23 and 24), as our reporting system is set up to accept files in this format.

The **miles1** script produces the delimited text file in the format discussed above. I'll explain changes you can make to the script to adapt it to your own needs after the listing.

MILES1 SCRIPT

```
#!/bin/ksh
##################################################################
#                                                                #
#       Written by:     Robert Russell                           #
#       Usage :         Builds milestone times files             #
#       Copyright :     Robert Russell (1998)                    #
#       Calls programs: calcdate_time                            #
#                                                                #
##################################################################

integer i
integer a
integer TOTAL_MIN_DIFF

COMMANDS=/usr/home/it032x/commands

CHECK_FILES="Build_Start:Build_1.fin:Build_2.fin\
:Build_3.fin:Build_4.fin:Build_5.fin:Build_6.fin\
:Build_7.fin:Build_8.fin"
MILESTONES=/usr/local/etc/milestones

MACHINE_ID="TPL"
FILE_EXT="Z"
SAVE_DIR=/usr/home/it032x/milestone
FIL_MON=`date +%b|tr "[a-z]" "[A-Z]"`

MONTH="00"
YEAR=`date +%Y`
DAY_DIFF="N"
TOTAL_MIN_DIFF=0

header_file()
{
integer h
h=2
while [[ `echo $CHECK_FILES|cut -f$h -d:`  != "" ]]
do
        TEMP=`echo $CHECK_FILES|cut -f$h -d:`
        HEADER=$HEADER"$TEMP,"
h=$h+1
done
echo "Key,Date,"$HEADER
}

month_name()
{
        case $1 in
        Jan)
```

```
                MONTH="01"
        ;;
        Feb)
                MONTH="02"
        ;;
        Mar)
                MONTH="03"
        ;;
        Apr)
                MONTH="04"
        ;;
        May)
                MONTH="05"
        ;;
        Jun)
                MONTH="06"
        ;;
        Jul)
                MONTH="07"
        ;;
        Aug)
                MONTH="08"
        ;;
        Sep)
                MONTH="09"
        ;;
        Oct)
                MONTH="10"
        ;;
        Nov)
                MONTH="11"
        ;;
        Dec)
                MONTH="12"
        ;;
        esac
}
#start main code
i=1
while [[ `echo $CHECK_FILES|cut -f$i -d:`  != "" ]]
do
        a=$i+1
        CHECK_ONE=`echo $CHECK_FILES|cut -f$i -d:`
        CHECK_TWO=`echo $CHECK_FILES|cut -f$a -d:`
        if [[ "$CHECK_TWO" != "" ]]
        then
                ls -al $MILESTONES/$CHECK_ONE|\
awk '{print $6,$7,$8}'|read ONE_MONTH ONE_DAY ONE_TIME
                ls -al $MILESTONES/$CHECK_TWO|\
awk '{print $6,$7,$8}'|read TWO_MONTH TWO_DAY TWO_TIME
```

```
                month_name $ONE_MONTH
                DATE_ONE=$ONE_DAY"/"$MONTH"/"$YEAR
                month_name $TWO_MONTH
                DATE_TWO=$TWO_DAY"/"$MONTH"/"$YEAR
                if [[ "$DATE_ONE" != "$DATE_TWO" ]]
                then
                        DAY_DIFF="Y"
                else
                        DAY_DIFF="N"
                fi
                        TOTAL_MIN=`$COMMANDS/calcdate_time\
 -d  $DATE_ONE $ONE_TIME $DATE_TWO $TWO_TIME`
                TIME_OUT=$TIME_OUT"$TOTAL_MIN,"
        fi
i=$i+1
done

FULLNAME=$SAVE_DIR/$MACHINE_ID$FILE_EXT$FIL_MON"A.TXT"
if [[ ! -f $FULLNAME ]]
then
        header_file > $FULLNAME
fi
grep $DATE_TWO $FULLNAME >/dev/null 2>&1
if [[ "$?" != "0" ]]
then
        print "k"`date +"%y%j"`","$DATE_TWO","$TIME_OUT>>$FULLNAME
fi
#end
```

To generate timings for your files, the lines of code detailed below must be changed. The list of files to be checked must be in both sequential (as implied by the file's name) and chronological order. The script won't function if files arrive in random order, only a sequential order enables the script to work correctly. In our scheme, the first file specified indicates the start of the batch run and the last one indicates the end of the run. This script can be used to calculate the timings between any set of 'marker files', which need not necessarily be batch milestone files – the process can deal with any set of files presented to it in sequential order.

Changes to implement miles1 script

```
COMMANDS=/usr/home/it032x/commands
```

The COMMANDS variable is used to execute the **calcdate_time** script, so its directory must be the same as both **calcdate_time** and

calcdate_diff.

```
CHECK_FILES="Build_Start:Build_1.fin:Build_2.fin\
:Build_3.fin:Build_4.fin:Build_5.fin:Build_6.fin\
:Build_7.fin:Build_8.fin"
```

The CHECK_FILES variable contains the list of milestone files to check. As mentioned above, these files must be listed and created in sequential order, as implied by their names (note that creating them in sequential order ensures they're also listed in chronological order).

```
MILESTONES=/usr/local/etc/milestones
```

The MILESTONES variable is where the files listed in CHECK_FILES are to be found. If your files are in separate directories, you'll need to modify code.

```
MACHINE_ID="TPL"
```

MACHINE_ID is the ID of the RS/6000 being checked. This value is also used in the naming of the results file.

```
FILE_EXT="Z"
```

The FILE_EXT variable stores a tag used in naming our results file. The one below is used in our set-up, and can be changed to suit your requirements (or, alternatively, entirely removed from the script).

```
SAVE_DIR=/usr/home/it032x/milestone
```

The SAVE_DIR variable is used to save the results file.

Implementation of this script and transfer of the resulting file into our reporting system allows us to produce a graphical representation of the batch run. This makes it easier for us to identify changes to the time it takes to run various parts of the batch process. A possible modification to the script would be to have it note start and end times in the results file so that reports of just this data are created. The DATE_ONE, ONE_TIME, DATE_TWO, and TWO_TIME variables could be added to the print statement for the results file (identified by the FULLNAME variable) at the end of the script to achieve this.

Milestone time generation

Our batch runs need to be verified at certain points to ensure they're on schedule, and this requires us to time-stamp certain events as they occur. To do this we generate ‘flag’ or ‘milestone’ files that the system expects and uses to monitor the progress of batch processing and activate the paging system, if necessary. In order to streamline the process of managing these milestone files we required a list of times throughout the night by which flag files should have been created. The following script was used to enable us to see the dates and times of these events. Automating this process has greatly improved it as it was previously very error-prone.

MILE_TIME SCRIPT

```
#!/bin/ksh
#set -x
################################################################
#
# Written by Robert Russell
#
# Produces a milestone time file based on the reference date and time
#
# The first positional parameter is the start date
# The second positional parameter is the start time
# The third positional parameter is the milestone jump in minutes
#
# eg  mile_times 01/01/1998 19:00  "205:15:15:420:30:240:330"
#
################################################################

integer POS
integer COUNT

COMMANDS=/usr/home/it032x/commands
CHECK=/usr/local/bin/Milestone_Lookout
OUT_FILE=/usr/home/it032x/data/MILE_TEMP

>$OUT_FILE
BASE_DATE=$1
ROTATE_DATE=$1
BASE_TIME=$2
PERIOD_JUMP=$3

if [[ "$3" = "-b"  ]]
        then
                STR=`grep "STR=" $CHECK|grep -v \
"#STR" |tail -2 |head -1|sed "s/\"//g"`
        fi
```

```
if [[ "$3" = "-w"  ]]
        then
                STR=`grep "STR=" $CHECK|grep -v \
"#STR" |tail -1|sed "s/\"//g"`
        fi

PRIN_DATE=`$COMMANDS/calcdate_diff -dc $BASE_DATE 0 -s -nn -de -mmm -yy`
print $PRIN_DATE" "$BASE_TIME" Start Time">>$OUT_FILE
COUNT=1
POS=1

while [ $COUNT -le 1 ]
do
        ROTATE_TIME=`echo $PERIOD_JUMP|cut -f$POS -d:`
        if [[ "$ROTATE_TIME" = "" ]]
        then
                POS=1
                COUNT=$COUNT+1
                ROTATE_TIME=`echo $PERIOD_JUMP|cut -f$POS -d:`
        fi
        TEMP=`$COMMANDS/calcdate_time -f \
$BASE_DATE $BASE_TIME $ROTATE_TIME`
        BASE_DATE=`echo $TEMP|awk '{print $1}'`
        BASE_TIME=`echo $TEMP|awk '{print $2}'`
        PRIN_DATE=`$COMMANDS/calcdate_diff \
-dc $BASE_DATE 0 -s -nn -de -mmm -yy`
        print $PRIN_DATE" "$BASE_TIME" after \
waiting $ROTATE_TIME mins">>$OUT_FILE
        ROTATE_DATE=$BASE_DATE
        POS=POS+1
done

more $OUT_FILE
#end
```

CHANGES NECESSARY TO IMPLEMENT MILE_TIME

The changes below are needed to adapt this script to suit your requirements.

```
    COMMANDS=/usr/home/it032x/commands
```

The COMMANDS variable is used to execute the **calcdate_time** and **calcdate_diff** scripts.

```
    OUT_FILE=/usr/home/it032x/data/MILE_TEMP
```

OUT_FILE is used as the results file.

Below is an example of this script being used from the command line:

```
mile_times 01/01/1998 19:00  "205:15:15:420:30:240:330"
```

This results in a file containing date and time entries starting at 19:00 on the 01/01/1998, with subsequent dates and times being determined by the delays (in minutes) specified by the third positional parameter. This is useful for calculating expected timing of events.

Period-end date calculation

This utility is similar to the one for milestone time generation, though it deals solely with dates. This process is used at our site to enable certain monitoring tasks to take place just before and just after period-end batch runs. As system activity during these periods is at its peak, we need to make a number of checks on our overall system status to ensure a trouble-free run. Our period-ends run on a ‘4, 4, 5-week’ cycle throughout the year. The following script calculates the period-end check dates. Developing scripts based on the **calcdate_diff** program has allowed us to schedule a number of checks automatically before each period-end. Further development has allowed us to monitor certain events that are supposed to happen in the second week of a period. The following script produces period-end dates based on a 4,4,5-week cycle.

PERIOD CHECKER SCRIPT

```
#!/bin/ksh
###############################################################
#
# Written by Robert Russell
#
# Produces a period date file for the reference date
#
# The first positional parameter is the reference date
# The second positional parameter is the period jumps in days
# The third positional parameter is the number of jumps
#
# eg  period_calculator 01/01/1998 "28:28:35" 4
#
###############################################################

integer POS
integer COUNT
```

```
COMMANDS=/usr/home/it032x/commands
OUT_FILE=/usr/home/it032x/data/PERIOD_DATES
FORMAT_DATE="-s -nnn -d -mmm -yy"

>$OUT_FILE
BASE_DATE=$1
ROTATE_DATE=$1
PERIOD_JUMP=$2
TIMES=$3
#PRIN_DATE=`$COMMANDS/calcdate_diff -dc $BASE_DATE 0 -s -nn -d -mmm -yy`
PRIN_DATE=`$COMMANDS/calcdate_diff -dc $BASE_DATE 0 $FORMAT_DATE`
print $PRIN_DATE>>$OUT_FILE
COUNT=1
POS=1

while [ $COUNT -le $TIMES ]
do
        DAYS=`echo $PERIOD_JUMP|cut -f$POS -d:`
        if [[ "$DAYS" = "" ]]
        then
                POS=1
                COUNT=$COUNT+1
                DAYS=`echo $PERIOD_JUMP|cut -f$POS -d:`
        fi
        if [ $COUNT -le $TIMES ]
        then
        BASE_DATE=`$COMMANDS/calcdate_diff -dc $ROTATE_DATE -$DAYS -/ -d
-m -yy`
        #PRIN_DATE=`$COMMANDS/calcdate_diff -dc $BASE_DATE 0 -s -nn -d -
mmm -yy`
        PRIN_DATE=`$COMMANDS/calcdate_diff -dc $BASE_DATE 0
$FORMAT_DATE`
        print $PRIN_DATE>>$OUT_FILE
        ROTATE_DATE=$BASE_DATE
        POS=POS+1
        fi
done

more $OUT_FILE
#end
```

The above script produces a text file, formatted using the **calcdate_diff** command, of period-end dates.

CHANGES NECESSARY TO IMPLEMENT PERIOD_CHECKER

```
COMMANDS=/usr/home/it032x/commands
```

The COMMANDS variable contains the location where the

calcdate_diff script is found.

```
    OUT_FILE=/usr/home/it032x/data/PERIOD_DATES
```

The OUT_FILE variable contains the name and directory that contains the results file.

```
    FORMAT_DATE="-s -nnn -d -mmm -yy"
```

The FORMAT_DATE variable is used to specify the format that the command should use for returning dates. The above example produces dates such as ‘Wednesday 12 August 1998’.

VARIOUS SCRIPTS

The following selection of scripts covers a variety of situations that commonly arise in data processing. In most cases, adapting the scripts to your needs requires only the COMMANDS variable to be changed (it contains the name of the directory where both the **calcdate_diff** and **calcdate_time** scripts are located). All the scripts below use **calcdate_time** and **calcdate_diff**, and serve as further examples of how these scripts can be used.

LAST DAY OF MONTH SCRIPT

This script checks whether the date in the first positional parameter is the last day of the month by comparing it to the following day.

```
#!/bin/ksh
#################################################################
#
# Written by Robert Russell
#
# Returns 0 if the date is the last day of month
# Returns 1 if the date is not the last day of month
#
# eg last_day_of_month 28/02/1996 returns 1
#
#################################################################

integer week

COMMANDS=/usr/home/it032x/commands

FIRST=`$COMMANDS/calcdate_diff -dc $1 0 -mm`
```

```
LAST=`$COMMANDS/calcdate_diff -dc $1 -1  -mm`

if [[ "$LAST" != "$FIRST" ]]
then
        print 0
else
        print 1
fi
#end
```

WEEK OF THE MONTH SCRIPT

This script takes the date in the first positional parameter and calculates the week of the month it's in, starting from the first day of the month.

```
#!/bin/ksh
######################################################
#
# Written by Robert Russell
#
# Returns the week number in the month
#
# Print the week number of the Date e.g. 1-5
#
# eg what_week_of_month 01/01/1997 returns "week one"
#
######################################################

integer week

COMMANDS=/usr/home/it032x/commands

FIRST=`$COMMANDS/calcdate_diff -dc $1 0 -mm`
week=1

for i in 7 14 21 28 35
do
LAST=`$COMMANDS/calcdate_diff -dc $1 $i  -mm`

if [[ "$LAST" = "$FIRST" ]]
then
        week=week+1
fi
done

case $week in
        1)
        print "week one"
        ;;
        2)
        print "week two"
        ;;
        3)
        print "week three"
        ;;
        4)
        print "week four"
        ;;
        5)
        print "week five"
        ;;
esac
#end
```

VALID DATE SCRIPT

This script can be used to process the return codes back from **calcdate_diff** script. The various return codes are listed in the script. Your own implementation may require further error handling than is provided here.

```
#!/bin/ksh
###############################################
#
# Written by Robert Russell
#
# Returns error messages for incorrect dates
#
###############################################

case $1 in
        1)
                print "Problem with year in date given"
                exit
        ;;
        2)
                print "Problem with month in date given"
                exit
        ;;
        3)
                print "Problem with day in date given"
                exit
        ;;
        *)
                print "Command ended in error"
                exit
        ;;
esac
#end
```

DAYS PASSED SINCE SCRIPT

This script calculates the number of days between the date given by the first positional parameter and the current system date. If a second positional parameter is used, it must be in 24-hour time format. If a time is specified, the output is in minutes.

```
#!/bin/ksh
##########################################################################
#
# Written by Robert Russell
#
# Calculates how many days or minutes have elapsed since
# a provided date
#
# Prints out the days/minutes
#
# Uses the current system date as the reference point
#
##########################################################################

COMMANDS=/usr/home/it032x/commands
CURRENT_DATE=`date +"%d/%m/%Y"`

if [[ "$2" = "" ]]
then
VALUE=`$COMMANDS/calcdate_diff -c $1 $CURRENT_DATE`
RETURN=$?

if [[ "$RETURN" != "0" ]]
        then
                $COMMANDS/valid_date $RETURN
        else
                print $VALUE
        fi
else
        CURRENT_TIME=`date +"%H:%M"`
        VALUE=`$COMMANDS/calcdate_time -d $1 $2 \
$CURRENT_DATE $CURRENT_TIME`
        print $VALUE
fi
#end
```

DAYS TO GO UNTIL SCRIPT

This script calculates the number of days from the current system date to the date given by the first positional parameter. If a second positional parameter is present, it must be in 24-hour time format. If the time is specified, the output is in minutes.

```
#!/bin/ksh
#####################################################################
#
# Written by Robert Russell
#
# Calculates how many days/minutes to go to a certain date
#
# Prints out the days/minutes remaining
#
# Uses the current system date as the reference point
#
#####################################################################

COMMANDS=/usr/home/it032x/commands
CURRENT_DATE=`date +"%d/%m/%Y"`

if [[ "$2" = "" ]]
then
        VALUE=`$COMMANDS/calcdate_diff -c $CURRENT_DATE $1`
else
        CURRENT_TIME=`date +"%H:%M"`
        VALUE=`$COMMANDS/calcdate_time \
-d $CURRENT_DATE $CURRENT_TIME $1 $2`
fi

print $VALUE
#end
```

CALCULATE WEEKS SCRIPT

This script calculates the number of weeks from the date given by the first positional parameter to the current system date, starting on week one.

```
#!/bin/ksh
####################################################################################
# Written by Robert Russell
#
# Calculates the number of weeks from a given date to the system date,
# starting on week one
#
###################################################################################

integer WEEKS
integer COUNT
integer TEMP
```

```
COMMANDS=/usr/home/it032x/commands
CURRENT=`date +"%d/%m/%Y"`
COUNT=`$COMMANDS/calcdate_diff -c $1 $CURRENT`
TEMP=COUNT/7
WEEKS=TEMP+1
print $WEEKS
#end
```

LAST WEEK OF THE MONTH

This script checks that the first positional parameter is within seven days of the end of the month. This could be changed to any number by changing the variable DAYS_LEFT.

```
#!/bin/ksh
##################################################
#
# Written by Robert Russell
#
# Returns 0 if the date is in the last n days of the month
# Returns 1 if it is not in the last n days of the month
# where n is given by DAYS_LEFT
#
##################################################

integer week
integer DAYS_LEFT

COMMANDS=/usr/home/it032x/commands
DAYS_LEFT=7

FIRST=`$COMMANDS/calcdate_diff -dc $1 0 -mm`
LAST=`$COMMANDS/calcdate_diff -dc $1 -$DAYS_LEFT  -mm`
if [[ "$LAST" != "$FIRST" ]]
then
        print 0
else
        print 1
fi
#end
```

Robert Russell (UK) © Xephon 1998

The benefits of 64-bit computing

The benefits of 64-bit computing are not always fully understood, and there are also some common myths about the performance of this new platform – I address both these issues in this article.

INTRODUCTION

The extraordinary rate of change in information technology, and the business pressure to use it for competitive advantage, places intense demands on information systems. Computing platforms must now be able to store and support a vast volume of information and make it available to an ever-increasing number of concurrent end-users. 64-bit departmental computing provides the latest means of securing a temporary competitive advantage to those organizations that are able to implement this new technology and integrate it with their existing systems in a relatively short time frame. In time, of course, even low-end desktop PCs will be 64-bit, just as a low-end PC now has a 32-bit Pentium processor that would have been the envy of just about every business user a few years back.

The PowerPC microprocessor architecture (which should not to be confused with a particular implementation of it, such as the one in PowerPC-based RS/6000 systems), supports both 32-bit and 64-bit implementations. 64-bit PowerPC implementations have a number of specific hardware features, including:

- 64-bit general purpose registers (GPRs).
- Dual execution mode, which allows the CPU to operate in either 32-bit or 64-bit mode.
- Instructions for operating on 64-bit data (load, store, logical, and mathematical) and for controlling the execution mode (32 or 64-bit).
- The ability to address physical memory larger than 4 gigabytes (the largest address that can be stored in 32 bits).

Doubling the number of bits from 32 to 64 does much more than double the number of possible values – this is a result of the exponential nature of the scale. Most people, I am sure, have heard the story about the inventor of the game of chess (whose name is now long forgotten). It's said that the king was so delighted with the new game that he asked the inventor what he could offer him as a sign of his gratitude. The wise, but anonymous, inventor replied that he wished for some corn – one grain on the first square, two on the second square, four on the third square, and so on, doubling the amount of corn on each square up to the 64th. The king, being a king, protested saying that this was insufficient for such a clever invention, but in the end accepted. Needless to say, the inventor lived happily ever after and was very, very rich, as the king was unable fully to pay his debt. The amount of grain required to fill the board is more than all the grain produced in the world since the beginning of time!

If we use distance to make the analogy and let one be equivalent to 10nm (or ten one-millionths of a millimetre), then the maximum number represented in 32 bits would be 43 metres (about 50 yards), while the distance represented by 64 bits would be more than the distance between the earth and the sun.

It's necessary to make clear the distinction between a *byte* and a *word*. While a byte is always a group of eight bits, a word is the unit of data that is natively handled by the CPU. On 32-bit processors, a word is 32 bits long, and on a 64-bit processor a word is 64 bits long. Note that this is the size of a word from the hardware's point of view; software may implement its own set of standards for handling data. For instance, Windows NT defines a 'double word' (as in the REG_DWORD data type) as four bytes or 32 bits – this is despite the fact that NT is a true 32-bit operating system. Word size is an important dimension in a computer architecture as it is used for two distinct purposes:

- Firstly, to represent data
- Secondly, to represent addresses.

While it's straightforward to handle 64-bit data on a 32-bit CPU, going beyond the 4 gigabyte address limit in a 32-bit architecture is a much more formidable problem.

It must be emphasized that most of today's applications don't come even close to testing the limits of 32-bit architectures and many run quite happily on 16-bit systems. Both Microsoft and Intel are still in the process of migrating from 16 to 32 bits, and the difficulties that this can present if not considered from the outset are only too apparent from the performance problems experienced by some 16-bit Windows applications running on 32-bit Intel processors. I expect a repetition of this problem when these two companies move to 64-bits, which, by all accounts, will not be before 1999 or 2000. Given that the PowerPC was designed from the outset to be a 64-bit architecture with a fully-functional 32-bit subset and a clear transition path, the move to 64-bits on AIX running on PowerPC processors should be much simpler.

Some applications are, however, running into the upper limits of 32-bit performance, and it is these that are driving the move to 64-bit data and addresses. These applications are those that handle large volumes of data. Over time, processors are becoming ever more powerful and more and more data is being stored in databases. On today's 32-bit octo-processor systems, a maximum of 512 MB of memory is available per processor (on average), which in many instances is barely enough to maintain a balanced system. If powerful CPUs are not to become I/O or memory bound, it is necessary that database memory buffer pools are allowed to grow so that the performance of existing systems isn't impaired. Buffer pools are even today reaching and exceeding the 4 GB limit imposed by 32-bit systems.

A 64-bit computer is one equipped with one or more processors capable of handling and manipulating 64-bit data and 64-bit addresses. On a 32-bit system, a C programmer would work with 32-bit integers (int), long integers (long), and pointers (addresses). This is known as 'ILP32' system. By agreement, 64-bit systems still use 32-bit integers, but long integers and pointers become 64-bit, giving rise to the 'LP64' system (notice the missing 'I').

The performance advantage of 64-bit hardware cannot be realized unless the system also uses a 64-bit operating system and runs applications designed to use 64-bit processing. This is the problem with Sun's Solaris – the underlying processor is 64-bit, but the operating system supports only 32-bit applications.

To summarize, the four distinguishing features of 64-bit systems by comparison with 32-bit systems are:

- Large virtual address space
- Large physical memory size
- Native 64-bit integer calculation
- Large files.

These features are discussed below.

LARGE VIRTUAL ADDRESSES SPACE

32-bit applications have a 4 GB (2^{32}) virtual address space. This means that the combined size of the binary executable and working data set cannot exceed 4 GB. Applications that work with more than 4 GB of data benefit from large quantities of real memory (RAM) and support for large virtual address spaces. 64-bit computing is most beneficial to these 'very large database' applications (VLDBs) – while just five years ago a database as large as 10 GB would have been rare, today it's commonplace. Today there are plenty of terabyte-sized databases around (a terabyte is about a million megabytes). It's worth bearing in mind that a database of up to 16 GB fits in the RAM of a fully configured Escala RL470; the performance improvement with 64-bit will be spectacular for such applications. There are no hard-and-fast rules to estimate just how much faster a database will run with a 64-bit architecture, though it's not unusual for applications to run from two to tens of times faster when RAM is increased from 4 to 16 GB.

A 64-bit application has a virtual address space of 16 EB (16 million TB ~ $18x10^{18}$ bytes), which is essentially limitless from the point of view of today's applications. This allows very large buffers to be used by databases, and their contents to be accessed directly by memory pointers. When combined with large physical memory, significant performance improvement can be achieved.

In addition to being able to access more data, the size of the application's binary image is freed from its 4 GB limit. Though not a problem today, the arrival of general purpose multi-processor machines and a standardized threading environment points the way to multi-threaded applications. In this case, the overall size of each process is likely to grow. Further, 64-bit applications use a 64-bit XCOFF binary format, which is somewhat larger than its 32-bit counterpart. This has the effect of increasing the size of the working set, which in turn reduces the efficiency of the cache hierarchy as the cache-hit ratio is reduced.

Mapping very large objects, such as file-systems and databases, directly into the virtual address space eliminates some of the overhead of address translation and also allows the application to benefit from hardware-based assist mechanisms for virtual-to-physical address translation.

LARGE PHYSICAL MEMORY

If the benefits of large virtual address spaces are to be realized, systems must also have large quantities of real memory, otherwise the system spends time paging virtual memory, which defeats the point of having a 64-bit architecture. With a large physical memory (over 4 GB), applications can keep most or all of their working data in RAM, thus eliminating costly I/O. However, the number of real-world applications that have a working set of more than 4 GB is comparatively small. Applications that use less than 4 GB will not experience the same improvement in performance as those that could access more memory, were it available.

Large physical memory also benefits 32-bit processes (those with a four gigabyte address space) by allowing several 32-bit processes to reside in main memory at the same time, eliminating or reducing the need for paging by the operating system.

Additionally, for a computer system to be efficient it must have ‘balanced’ resources. This means that the overall performance of a computer depends on physical memory, CPU, and I/O bandwidth. If you increase one of them you should increase the other two by a similar amount. Improvements in microprocessor performance have not been matched by similar improvements in memory and I/O. Additional memory helps improve the balance.

NATIVE 64-BIT COMPUTING

Handling integers larger than 32 bits on 32-bit systems generally requires a library call. A 64-bit processor with a 64-bit word size can manipulate and operate on 64-bit long data directly, eliminating the library call. This feature is useful not only for scientific and technical applications but also for commercial applications that perform 64-bit arithmetic on 64-bit addresses.

LARGE FILES

Even though files larger than 4 GB can be handled by AIX 4.2, AIX 4.3's 64-bit support makes this more 'native', as files are no longer divided into 2 GB or 4 GB chunks. This is a direct consequence of a large virtual address space.

COMPATIBILITY – THE CUSTOMER'S REQUIREMENT

There are over 14,000 32-bit applications that run on AIX 4.2 today. It is essential that these applications should continue to run unmodified on 64-bit platforms under AIX 4.3. With this new version of the operating system, both 32 and 64-bit applications may run side-by-side and communicate and exchange data with each other. Large physical memory may benefit applications comprising a number of 32-bit processes, as they and all their data can reside in real memory, and not be paged to disk.

This is a direct benefit of the PowerPC architecture (see Figure 1), which was conceived as a 64-bit architecture from the outset, with a 32-bit mode of operation fully defined and support for binary compatibility between the two. This means that 32-bit applications display exactly the same behaviour when running on 64-bit hardware as on 32-bit hardware. This greatly simplifies the transition from 32 to 64-bit architectures. For application developers, the AIX compiler provides a compilation mode that produces binaries that will run on any PowerPC, POWER, or POWER2 processor.

AIX 4.3 also supports both 32 and 64-bit computing. There is only one version of the operating system, which runs on a wide range of hardware. Once more, for an application to enjoy the benefits of 64-

bit computing it must be compiled in 64-bit mode and linked to 64-bit objects in shared libraries.

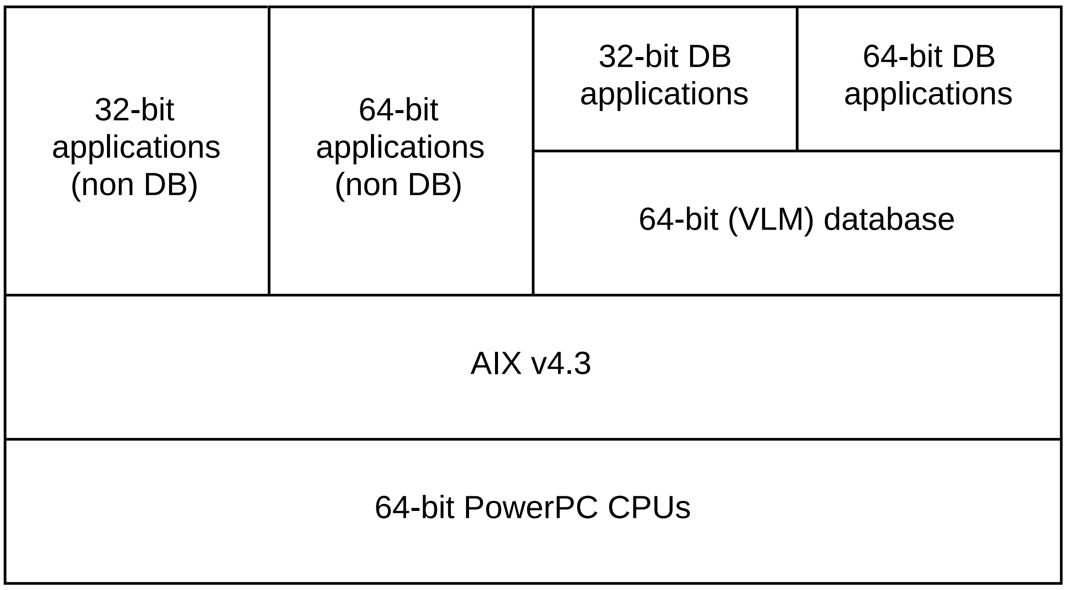

Figure 1: 64-bit PowerPC architecture

MIXING 32 AND 64-BITS

There are a few rules that need to be taken into consideration when mixing 32 and 64-bit hardware and software. Firstly, it isn't possible to run a 64-bit application on 32-bit hardware – 64-bit applications require 64-bit registers, etc. Secondly, while apps built for AIX v3 and versions of AIX v4 prior to 4.3 generally run on AIX v4.3, there's no guarantee that the converse is true.

Other than differences in addressability, there are a number of differences between 32-bit and 64-bit execution environments. The sizes of the following are architecture-dependent (for instance, having 64-bits on a 64-bit machine):

- Pointers types
- The C language's 'long' type
- CPU registers.

Additionally, there are a number of PowerPC instructions that are specific to 64-bit implementations of the architecture.

32-bit applications making use of 64-bit databases will not require any modification, as the interfaces to the database remain 32-bit. The mapping from the application's 32-bit environment to the database's 64-bit environment is handled by the API.

A number of development tools have been modified to generate warnings when architecture-specific code is encountered (they specifically look for code that would make a transition from 32 to 64-bits more difficult). These tools include **lint**, **ar**, **strip**, **ranlib**, and **dump**.

THE BENEFITS OF 64-BIT APPLICATIONS

Large database applications are the real beneficiaries of the arrival of 64-bit processors. These applications can make the most of the very large physical memory and virtual address space that 64-bit systems offer. Savings in execution time that fully configured 64-bit systems offer in relation to 32-bit systems means that, in general, the extra cost of the larger system is quickly repaid. Applications that make use of large database technology are becoming increasingly common in Enterprise Resource Planning (ERP), on-line transaction processing (OLTP), and data warehousing/decision support (DW/DSS).

Oracle, Sybase, and Informix all have experience of developing and operating VLDBs, and all provide database engines that can utilize more than 4 GB of real memory. The discussion that follows concentrates on Oracle's product, though the principles discussed apply equally to products from all vendors.

Oracle's 64-bit database is called OracleVLM (Oracle Very Large Memory). It uses two key architectural components:

- Large System Global Areas (SGAs)
- 'Big Oracle Blocks'.

Oracle data is stored in system memory in the System Global Area. Increasing the size of the SGA reserves more contiguous shared memory. The larger the Oracle buffer cache, the faster the Oracle RDBMS runs. SGA comprises four areas:

- Fixed size
- Variable size
- Database buffers
- Redo buffers.

OracleVLM has enlarged the database and redo buffers to support up to 4 GB each, so the database may have more than 8 GB of buffers.

32-bit operating systems are limited to a maximum of 4 GB of system memory, which means that the size of the SGA the database system can use is similarly limited. The very large SGAs that are available with OracleVLM allow many more database blocks to be cached, improving the cache hit ratio.

Big Oracle Blocks, or BOBs, allow disk blocks on the Oracle file system to be up to 32 KB in size. This has a number of advantages: firstly the overall data transfer rate between disk and memory is improved as a result of fewer seek operations. Secondly, larger blocks result in reduced fragmentation in the database's memory, which means more rows per block and a less complex data layout. It should be noted that BOBs, which are smaller than 4 GB, do not require 64-bits.

Databases on their own are not much use – all they do is hold data. To be of any value, they require a supporting application. In the commercial world there are three main types of database application:

- On-line transaction processing (OLTP)
- Data warehousing/decision support systems (DW/DSS)
- Enterprise resource planning (ERP).

TRANSACTION PROCESSING

Transaction processing is the system that is generally used for the day-to-day running of a business. The often-cited example is that of a credit card purchase, in which the purchaser's bank account is debited by the appropriate sum and the seller's account credited by the same amount. In this case, the transaction either fails or succeeds. It cannot 'partly succeed', so that, for instance, the vendor's account is credited but the buyer's account isn't debited. Another everyday example of transaction processing is ticket and hotel reservation. The transactions themselves usually involve relatively small amounts of data, typically 4 KB, and access data in the databases in fairly random patterns with about two-thirds of the accesses being read operations.

DATA WAREHOUSING/DECISION SUPPORT

Data warehousing or decision support systems (DSS) are used to analyse the historical data collected by an organization, often with the objective of finding hidden relationships. The results of this analysis and any relationships found can help the organization's strategic planning. Examples of DSS include examining the effect of advertising or the weather on sales.

The database operations involved in data warehousing are very different from those of transaction processing. A typical request may have a simple semantic, such as "How many more red swimming trunks did I sell in August in the shop in Paris, when it was sunny outside, and after the price reduction?" To produce the answer, the data warehouse application generally scans all the databases from start to finish, then performs an operation called a join to produce a new database table containing the results. From this, it can be seen that the amount of data read is huge, and that the accesses are generally sequential. Further, with this type of application a large percentage of the data is not re-used – it is just read and thrown away.

ERP

Enterprise resource planning is also an operational system, in that it is used to manage the day-to-day functioning of an organization. The best-known packages are SAP R3, Baan IV, Peoplesoft, and Oracle applications. The size of the databases is generally somewhere between that of an OLTP database and one used by a DSS. The nature of the operations performed on the databases is very similar to those of OLTP. A key difference between OLTP and ERP is that, with the latter, there is a great deal of interaction between the different tables in the database. Very large memory means that these frequently used tables can be held in RAM as opposed to on disk.

APPLICATION TYPE AND 64-BIT PROCESSING

Despite the very different nature of these three types of database application, very large memory offers the same type of advantage to each. 64-bit addressing allows applications to cache large parts of or even the whole database in memory. However, as OLTP databases are generally smaller than those used for data warehousing and ERP, a higher percentage of the data is held in RAM, and the cache-hit ratio is better. Thus, while large real memories improve all three types of application, the performance increase for transaction processing is more marked than that for data warehousing and ERP.

All database engines perform a number of optimizations on transactions and queries run against the database. Normally, these optimizations consider memory usage, access time, and I/O load. With VLM databases, the latter part of the equation can be ignored. This greatly simplifies the optimization phase, making it quicker and more efficient.

Obviously, all the data must be loaded from disk into system memory at the outset. As with all cache architectures, if there is no re-use of the data, there is no benefit. Because of this and the size of datasets typically used in decision support applications, a relatively poor cache hit ratio is common, and the performance improvement in using 64-bit processing is considerably less than for datasets that fit completely inside database buffers.

64-BIT MYTHS

There are two false ideas that are often associated with 64-bit computing, these are:

- 64-bit binaries are twice as big as 32-bit ones
- All 64-bit applications run faster than 32-bit ones.

‘64-bit binaries are twice as big as 32-bit ones.’

While it’s true that 64-bit binaries are larger than 32-bit ones, they are a long way from being twice as big. A 64-bit application has a 64-bit XCOFF binary format. This means that addresses and ‘long ints’ are represented as 64-bits. But as PowerPC instructions are 32-bits long and the standard convention for the C language type ‘int’ is 32-bits, and ‘char’ is 8-bits (even on a 64-bit processors) not all objects in the XCOFF file double in size.

‘All 64-bit applications run faster than 32-bit.’

This derives, I am sure, from the ‘bigger is better’ concept, which doesn’t always hold true. An application compiled in 64-bit mode only runs faster than the same code compiled in 32-bit mode if it makes use of one or more of the features of 64-bit systems, such as very large memory or native handling of 8-byte data. The reasons for this are given below.

- A large virtual address space means that larger applications can be developed and executed, but this has no relation to performance.
- Support for large physical memory doesn’t result in a performance improvement if the extra physical memory is not present. In general, the greater the amount of physical memory the less the operating system will perform paging. This results in better performance, but it is due more to software than hardware.
- A 64-bit processor can perform 64-bit arithmetic. However, if an application only uses 32-bit data, arithmetic will not be performed twice as fast on a 64-bit processor – typically, it takes the same number of processor cycles irrespective of the PowerPC model. If, however, an application uses 64-bit data then there is a performance improvement, as the manipulation of 64-bit data on a 32-bit processor requires library calls, which is significantly slower than native 64-bit operations.
- Large file and filesystem support simplifies the manipulation of large datasets. It’s possible to map these large files into memory, but again, unless the additional physical memory is available, little or no performance increase results. Paging, on the other hand, is a software issue, not a hardware one.

Additionally because 64-bit binaries are slightly larger than their 32-bit counterparts, they have a poorer cache hit ratio. The cache hit ratio plays an important part in system performance, so it is possible that a 32-bit binary may run slower when compiled and run in 64-bit mode. The cache hit ratio can be recovered by increasing the size of the processor and system caches, but, because of the cost and power consumption of static RAM, this is an expensive route to take.

Because of the above, it is probable that only database applications will be re-compiled in 64-bit mode. It will probably not even be necessary to re-compile the applications that make use of the database, such as the transaction processing monitor and the decision support application. So, the golden rule is that, if your application doesn't need 64-bits, leave it alone. The default mode for development tools is to create 32-bit objects and applications.

CONCLUSION

For commercial information technology systems the advantages of 64-bit computing are large physical memories and large virtual address spaces. With 64-bit addressing, applications can cache large parts or the whole database in main memory: 64-bit computing can produce dramatic performance improvements and reduced response times. Applications that have a working set larger than 4 GB, which is the address limit of 32-bit systems, will experience significant performance improvements when re-compiled in 64-bit mode and executed on 64-bit systems that have a large physical memory. Applications that do not require 64-bit processing should not be changed.

With the release of AIX v4.3, 64-bit Unix is now available on a 64-bit hardware architecture. Additionally, AIX v4.3 brings 64-bit support and offers compatibility with existing 32-bit applications running on earlier versions. The intention behind support for 32-bit applications is simple – to provide a smooth transition from 32-bit to 64-bit.

Jez Wain
Bull Information Systems (France)

Disk management

In AIX 4.1.4 and 4.2.*x*, it is not necessary to vary on the volume group – all you have to know is one of the physical *hdisk* names of the volume group. If you take over a volume group, and don't have all the required logical volume and physical volume information in hardcopy, you can use the **lqueryvg** command to gather information about the volume after physical volumes are connected to the target system without importing the volume group.

There are many other situations in which this command is also helpful. For instance, under the section headed 'Physical', the physical volume identifier is displayed.

```
lqueryvg -p hdisk73 -At

Max LVs:        256

PP Size:        25

Free PPs:       100

LV count:       6

PV count:       4

Total VGDAs:    4

Logical:        000b345d5b164cab.1   oracle 1

                000b345d5b164cab.2   arch 1

                000b345d5b164cab.3   userdump 1

                000b345d5b164cab.4   log 1

                000b345d5b164cab.5   redo 1

                000b345d5b164cab.6   loglv00 1

Physical:       00089f9cd4602141 1   0

                00089f9cd4602700 1   0

                0008823d3e8e478b 1   0

                0008823d3e8e04e4 1   0
```

Michael Imhotep (Australia) © Xephon 1998

Getting rid of verbose error messages

Many of the messages returned by Unix commands are available in different languages. This is controlled by the *LANG* environment variable, the default being *En_US* meaning ‘English native to the US’. All the default messages have a message number associated with them, for example:

```
$ cat no_such_file

cat: 0652-050 Cannot open no_such_file.
```

If you prefer more terse, Unix-like error messages, set the environment variable *LC_MESSAGES* to *C*, and you will get:

```
$ cat no_such_file

cat: Cannot open no_such_file.
```

By default *LC_MESSAGES* is the same as the *LANG* environment variable. Setting the *LANG* environment does the same trick, but should be avoided as it also changes application defaults.

AIX Specialist © Xephon 1998

How to fsck the root filesystem

You can run **fsck** in either maintenance mode or on mounted filesystems.

1 Boot from diskette (AIX 3), CD, or tape (AIX 4).

2 Select maintenance mode.

3 Type **/etc/continue hdisk0 exit** (replace *hdisk0* with the name of the boot disk, if it’s not *hdisk0*).

4 Issue the command: **fsck /dev/hd4**.

AIX Specialist © Xephon 1998

AIX news

BEA Systems has announced the M3 object transaction manager along with a gaggle of related products and partnerships with hardware, software, and systems integration firms. M3 is an object-oriented version of the company’s Tuxedo TP monitor, and promises Tuxedo’s scalability, load balancing, performance, and other functions and management tools.

M3 is out now on AIX, Digital Unix, HP-UX, NT (Intel and Alpha), and Solaris, and costs the same as Tuxedo.

For further information contact:
BEA Systems, 385 Moffett Park Drive, Sunnyvale, CA 94089, USA
Tel: +1 408 743 4000
Fax: +1 408 734 9234
Web: http://www.beasys.com

BEA Systems Ltd, Windsor Court, Kingsmead Business Park, Frederick Place, London Road, High Wycombe, Bucks HP11 1JU, UK
Tel: +44 1494 559500
Fax: +44 1494 452202

* * *

Candle has announced the Roma Business Services Platform (Roma BSP) version 1.1, which integrates component, message queuing, and directory technologies. The company also announced Roma for e-business, along with a GUI front-end for the product.

BSP runs on AIX, Solaris, MVS, HP-UX, NT, and AS/400. BSP 1.1 for AIX, NT, and Solaris is out now, costing from US$8,500 for a starter kit, with additional copies costing US$750 per seat.

For further information contact:
Candle, 2425 Olympic Boulevard, Santa Monica, CA 90404, USA
Tel: +1 310 829 5800
Fax: +1 310 582 4287
Web: http://www.candle.com

Candle Service Ltd, 1 Archipelago, Lyon Way, Frimley, Camberley, Surrey GU16 5ER, UK
Tel: +44 1276 414700
Fax: +44 1276 414777

* * *

Sybase has announced Warehouse Studio, an integrated product set for building departmental or enterprise data warehouse applications. It includes a set of design, transformation, database, meta data management, and administration facilities.

Prices for the AIX version start at US$76,200.

For further information contact:
Sybase Inc, 6475 Christie Avenue, Emeryville, CA 94608, USA
Tel: +1 510 922 3500
Fax: +1 510 658 9441
Web: http://www.sybase.com

Sybase (UK) Ltd, Sybase Court, Crown Lane, Maidenhead, Berks SL6 8QZ, UK
Tel: +44 1628 597100
Fax: +44 1628 597000