39

AIX

January 1999

update

In this issue

Published by

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

North American office

Xephon/QNA
1301 West Highway 407, Suite 201-405
Lewisville, TX 75077-2150
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

Disclaimer

Readers are cautioned that, although the information in this journal is presented in good faith, neither Xephon nor the organizations or individuals that supplied information in this journal give any warranty or make any representations as to the accuracy of the material it contains. Neither Xephon nor the contributing organizations or individuals accept any liability of any kind howsoever arising out of the use of such material. Readers should satisfy themselves as to the correctness and relevance to their circumstances of all advice, information, code, JCL, scripts, and other contents of this journal before making any use of it.

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).

Printed in England.

File archives and compression

This article discusses why small files may be just as large as large files, why one file may be better than two, and how to squeeze more space out of your disk. It delves into some of the technical aspects of disk storage, but I have tried to cover them in a non-technical way.

To the technocrat, I apologize for anything that may appear as an over-simplification of the process, but I am not trying to teach disk operating system design theory here. This article is designed to provide a simple overview of disk space usage.

HOW BIG IS A FILE?

You would think that the answer to this question is simple: just type **ls -l** and the fifth column of the directory listing tells you how many bytes are in each file in a directory. In the example listing below *meeting.txt* is three bytes long (a very short meeting) and *note.txt* is 1201 bytes long.

```
$ ls -l
total 6
-rw-r--r--   1 mjb     group        3 Aug 02 23:31 meeting.txt
-rw-r--r--   1 mjb     group     1201 Aug 02 23:25 note.txt
```

While these are the file sizes, they don't reflect the amount of space used by each file on the disk. To see the space used by each file, use the **-s** switch. Type **ls -ls**. This adds a new column at the front of each directory entry that contains the number of blocks used by the file. A block is a unit of 512 bytes. The first file, *meeting.txt*, uses 2 blocks, or 1024 bytes, (maybe it wasn't such a short meeting after all!) and *note.txt* uses 4 blocks, or 2048 bytes.

```
$ ls -ls
total 6
   2 -rw-r--r--   1 mjb   group      3 Aug 02 23:31 meeting.txt
   4 -rw-r--r--   1 mjb   group   1201 Aug 02 23:25 note.txt
```

It would be nice if a three-byte file used exactly three bytes plus some bytes for the file name and other information in the directory. Unfortunately that has never been a practical way to organize a disk.

The overhead in keeping track of the directory would become a load on the system. As a file expands or contracts as a result of editing or data entry, the file would become heavily fragmented. The first three bytes of the original file would be down on track 12, then the next fourteen bytes of data you added would be over on track 25, and so on. The more you add to the file, the more it gets fragmented. The directory would become a jumble of pointers, pointing to little bits of file all over the drive. Loading a file into the editor would involve the disk read heads scrambling over all the disk to pick up directory information and all the little pieces of the file for assembly into one large stream of bytes.

To handle this scattering, disk organization is a compromise. The method used by all major operating systems in one form or another is to select a convenient number of bytes as the minimum storage that can be allocated to a file. This amount could be called an 'allocation unit' – some operating systems do call it an allocation unit, but the terminology varies. When a file is first created and new data is added to it, it is allocated a sufficient number of allocation units to hold all its data. Most operating systems attempt to allocate this space as a series of contiguous sectors on the disk. If the file does not use all the space in an allocation unit, the end of the file is stored at the beginning of the last allocation unit and the remainder of the allocation unit is set aside for further expansion of the file. When a file expands, the new information is added to the end of the free space at the end of the last allocation unit that is allocated to the file. If the size of the information does not exceed the number of bytes in an allocation unit, all the new information is stored in the empty reserved space. Once additional data exceeds the free space in the allocation unit, another allocation unit is grabbed and reserved. The spill over from the allocation unit that overflowed is tucked at the start of the new allocation unit, and so on.

This leaves directory and file location logic with two simpler problems:

1 Given a file name in a directory, how do you locate the first allocation unit of the file?

2 Given the first allocation unit for a file (or any allocation unit), what method is used to chain the allocation units together?

A discussion of chaining methods is beyond the scope of this article, but they basically use pointers, stored somewhere in the allocation unit or the directory structure, to point to the next allocation unit. A disk file is read by reading the directory, picking up the first allocation unit, reading through that section of the disk, and locating the pointer to the next allocation unit and proceeding to that portion of the disk to retrieve the next chunk of file.

Earlier Unix systems used an allocation unit of 512 bytes. This came to be known as a 'block'. As disk capacity grew, the basic allocation unit was increased to 1,024 bytes on most systems (larger on some), but many utilities, such as **ls**, report file sizes and disk use in 512-byte blocks. This block size has remained the standard for many utilities, even though the size of an allocation unit may have increased to two or four blocks.

The allocation unit and the physical sectors on a disk are related. An allocation unit always comprises one or more physical disk sectors. Disk sectoring is yet another subject, and it relates to the physical layout of the disk. Blocks and allocation units are logical ways of grouping one or more sectors together.

Let's look at the **ls -ls** listing again. A three-byte file, like *meeting.txt*, will occupy three bytes of a 512-byte block, but (more importantly) it will occupy three bytes of one allocation unit. On the system in the example below, the allocation unit is two blocks or 1,024 bytes. The **ls -ls** listing correctly indicates that two blocks are used on the disk. Similarly, *note.txt* is 1,201 long and should occupy two full blocks and part of a third (1,024 bytes plus an additional 177 bytes in a third block). Hence *note.txt* uses two allocation units, or four blocks, as indicated in the listing.

```
$ ls -ls
total 6
   2 -rw-r--r--   1 mjb   group      3 Aug 02 23:31 meeting.txt
   4 -rw-r--r--   1 mjb   group   1201 Aug 02 23:25 note.txt
```

This seems wasteful. In fact 99.7% of the space allocated to *meeting.txt* is unused, as is 41.4% of that allocated to *note.txt*. Multiply this by the number of files on the system and it's easy to imagine vast black holes of disk space that cannot be filled except by forcing all users to create and use only files that occupy multiples of 1,024 bytes.

Before you throw a fit, remember that high percentage waste occurs only with very small files – larger files use their storage allocation much more efficiently. Your system is probably working fairly well, which is testimony to the fact that the allocation unit method is a good compromise between disk allocation and speed of disk access.

One useful task is to establish what is the allocation unit size of your system. You could find this in a manual (if you have a couple of hours to search for and through them), though a simpler method is to read the **man** page for **ls**. Establish the block size used by the **-s** option (usually 512 bytes), then use **vi** to create a file with only few a bytes in it and close the file. Use **ls -ls** and look at the number of blocks used to store that small file, multiply it by the block size, and you now know your basic allocation unit size.

Another useful fact: the **ls -l** and **ls -s** commands (and other similar variations) display a 'total' line as the first record in a directory display. The 'total 6' in the listing below is in fact the sum of the blocks displayed by the **ls -ls** command.

```
$ ls -ls
total 6
   2 -rw-r--r--   1 mjb   group      3 Aug 02 23:31 minutes.txt
   4 -rw-r--r--   1 mjb   group   1201 Aug 02 23:25 note.txt
```

Now that you've identified some files that are consuming an unnecessary amount of disk space, what can you do about it? You may have tried using one or more of the various file compression utilities (**pack**, **compress**, and the GNU Software utility **gzip**) on them. These utilities work well on large files, but perform poorly on small files. In the sample listing below, **compress** is applied to each of the files and the results are displayed. **compress** correctly recognizes that it can't do any good to *meeting.txt*, and leaves it uncompressed, so it just compresses *note.txt* to 188 bytes. Note that **compress** appends *.Z* to a file name after it compresses the file. The effects of **compress** are reversed by using **uncompress**, or **compress -d *file.ext***. You do not need to include the *.Z* in the file name.

In this case we have eliminated two blocks, as *note.txt* compressed down to two blocks from four. If you consider it, you'll see that a small file can never be compressed below two blocks (or the default allocation unit of your system).

```
$ ls -ls
total 6
   2 -rw-r--r--   1 mjb   group      3 Aug 02 23:31 meeting.txt
   4 -rw-r--r--   1 mjb   group   1201 Aug 02 23:25 note.txt
$ compress minutes.txt
$ compress note.txt
ls -ls
total 4
   2 -rw-r--r--   1 mjb   group      3 Aug 02 23:31 meeting.txt
   2 -rw-r--r--   1 mjb   group    188 Aug 02 23:25 note.txt.Z
$
```

If you have a directory of small files that are little used, but need to remain on the system, one way to handle them is to combine them into one file, then remove the originals. If the files can be 'strung together' then all the little files could be packed into one larger file. The **tar** (tape archive) utility is one obvious candidate for this, so we'll try it.

The **tar** command uses letters to indicate the actions to be performed. These are a bit like command-line switches, but are not preceded by a '**-**'. In the example listed later on, the **tar** arguments used are:

c	Create a new archive
v	Provide verbose information on what you are doing
f	The next argument is the name of the archive to create
txt.tar	The archive being created
**.txt*	The list of files to include in the archive

Immediately after you issue the **tar** command, **tar** informs you that it has appended *meeting.txt*, which would take one tape block.

```
a meeting.txt 1 tape block
```

It also informs you that it has appended *note.txt*, which takes 3 tape blocks.

```
a note.txt 3 tape blocks
```

So **tar** reports its results in 512-byte blocks rather than 1,024-byte double blocks.

However, there's a bit of a shock in store when you issue the **ls -ls** command after **tar** is complete. The new archive, *txt.tar*, is eight

blocks long. That's longer than the original six blocks used by the two files. The **tar** utility is a bit mindless. It doesn't actually string files end-to-end, but strings blocks end-to-end. It also has to add directory information into *txt.tar*, so it is not unusual (in fact common) for **tar** archives to be larger than the sum of their contents!

```
$ ls -ls
total 6
   2 -rw-r--r--   1 mjb   group      3 Aug 02 23:31 minutes.txt
   4 -rw-r--r--   1 mjb   group   1201 Aug 02 23:25 note.txt
$ tar cvf txt.tar *.txt
a minutes.txt 1 tape block
a note.txt 3 tape blocks
$ ls -ls
total 14
   2 -rw-r--r--   1 mjb   group      3 Aug 02 23:31 minutes.txt
   4 -rw-r--r--   1 mjb   group   1201 Aug 02 23:25 note.txt
   8 -rw-r--r--   1 mjb   group   4096 Feb 05 01:40 txt.tar
$
```

This makes things look a little desperate. According to the manual entry on **tar**, the unused space at the end of each block is filled with garbage. In fact this garbage is usually a lot of hexadecimal zeroes or *NUL*s, which makes a **tar** archive an excellent candidate for compression.

Proceeding to the next logical step, the following listing compresses the **tar** archive. The resulting file *txt.tar.Z* is 404 bytes long (one allocation unit or two blocks). By removing the original text files, the directory contents are reduced to only two blocks, a saving of 66% of the space previously used.

```
$ ls -ls
total 14
   2 -rw-r--r--   1 mjb   group      3 Aug 02 23:31 meeting.txt
   4 -rw-r--r--   1 mjb   group   1201 Aug 02 23:25 note.txt
   8 -rw-r--r--   1 mjb   group   4096 Aug 02 01:40 txt.tar
$ compress txt.tar
$ ls -ls
total 8
   2 -rw-r--r--   1 mjb   group      3 Aug 02 23:31 meeting.txt
   4 -rw-r--r--   1 mjb   group   1201 Aug 02 23:25 note.txt
   2 -rw-r--r--   1 mjb   group    404 Aug 02 01:40 txt.tar.Z
$ rm *.txt
$ ls -ls
total 2
```

```
   2 -rw-r--r--   1 mjb   group    404 Aug 02 01:40 txt.tar.Z
$
```

The following listing shows you all the steps necessary to reverse the **tar** and **compress** process. The **tar** argument for extracting from an archive is **x**. The other arguments are the same as the earlier **tar** command.

```
$ ls -ls
total 2
   2 -rw-r--r--   1 mjb   group    404 Aug 02 01:40 txt.tar.Z
$ uncompress txt.tar
$ ls -ls
total 8
   8 -rw-r--r--   1 mjb   group   4096 Aug 02 01:40 txt.tar
tar xvf txt.tar
x minutes.txt, 3 bytes, 1 tape block
x note.txt, 1201 bytes, 3 tape blocks
$ ls -ls
total 14
   2 -rw-r--r--   1 mjb   group      3 Aug 02 23:31 meeting.txt
   4 -rw-r--r--   1 mjb   group   1201 Aug 02 23:25 note.txt
   8 -rw-r--r--   1 mjb   group   4096 Aug 02 01:40 txt.tar
$ rm txt.tar
$ ls -ls
total 6
   2 -rw-r--r--   1 mjb   group      3 Aug 02 23:31 meeting.txt
   4 -rw-r--r--   1 mjb   group   1201 Aug 02 23:25 note.txt
```

You can use **tar** and **compress** to save lots of disk space on files that are rarely used. The GNU Software **gzip** utility is more efficient than **compress**, especially on large files, and has some other options that make it more efficient than **compress**.

Once you've located a stash of files that are rarely used but need to be available, you should archive and compress them. For directories with many small files, use **tar** and **compress** or **gzip**. For directories with a few large files, **compress** or **gzip** them, you may **tar** them if you like, but it probably makes little difference to the amount of space used. It might be easier to administer to keep the process the same for any directory.

Mo Budlong (mobudlong@aol.com)
President
King Computer Services (USA)

A function to validate users on AIX

The ready-to-compile function *ValidUser()* is useful when you need to validate a userid/password pair on AIX. It has been tested on AIX 4.2. The process that uses the *ValidUser()* function must be running with *root* permissions. The function also works in conjunction with an NIS server.

VALIDUSER()

```
/*
** valuser.h
**
** UserID Authentication
**
*/

#define USER_OK   0
#define USER_KO   1

int ValidUser (char *szUserid, char *szClearPasswd, struct passwd
*pPasswdStruct);

/*
** valuser.c
**
** UserID Authentication
**
*/

/*
 *--------------------------------------------------------------------
 * include section
 *--------------------------------------------------------------------
 */
#include <pwd.h>
#include <stdlib.h>
#include <stddef.h>

#include "valuser.h"

/*
 *--------------------------------------------------------------------
 * ValidUser - Execute [userid,password] validation
 *
 * Arguments
```

```
 *     szUserid         - Userid
 *     szClearPasswd    - Password (not encrypted) to be validated
 *     pPasswdStruct    - Pointer to passwd structure corresponding
 *                        to szUserid (NULL if there is none or if
 *                        szUserid does not match a valid user)
 * Return
 *     USER_OK  if [szUserid, szClearPasswd] is a valid couple
 *     USER_KO  if [szUserid, szClearPasswd] is not a valid couple
 *-------------------------------------------------------------------
 */
int
ValidUser ( char *szUserid,
            char *szClearPasswd,
            struct passwd *pPasswdStruct )
{
    int  nReenter;
    char *sMsg;
    struct passwd *pw_buf;

    /*
    ** Authentication via 'authenticate' routine
    */
    if ( authenticate(szUserid,szClearPasswd,&nReenter,&sMsg) )
    {
        /*
        ** Zero the pPasswdStruct structure and exit
        */
        memset(pPasswdStruct,0x0,sizeof(struct passwd));
        return (USER_KO);
    }
    else
    {
        /*
        ** Read struct passwd record from /etc/passwd for szUserid
        */
        setpwent();
        pw_buf = getpwnam(szUserid);
        memcpy(pPasswdStruct,pw_buf,sizeof(struct passwd));
        endpwent();
        return (USER_OK);
    }
}
```

Marco Pirini
System Administrator (Italy)

Understanding the sort command

AIX users may need to display text in a format different from the way it's stored in a file. For example, they may want to display a table alphabetically, or they may need to display numbers in a database in numerical order. The **sort** command can accomplish this.

SORT COMMAND BASICS

Figure 1 shows the basic syntax of the **sort** command. **sort** takes as its input a specified file (or, in certain cases, an input device), transforms the input, and writes the results to the screen, output file, or another device. The result of the **sort** command is typically all the lines in the file sorted in the order specified on the command.

For example, suppose you have a file called *this.text* that contains the following lines:

```
cards
balloons
apples
dogs
```

If you were to enter the command **sort this.text**, the results would be:

```
apples
balloons
cards
dogs
```

In this case, the command simply performs an alphabetic sort on the file. If multiple files are specified, **sort** concatenates the files and then sorts the resulting file.

SYNTAX OF THE SORT COMMAND

```
sort options filespec

   options   an optional flag or flags used to enhance the sort
             operation

   filespec  the file upon which the sort operation is to perform
```

The **sort** command is frequently used to display data found in tables and databases. Flat files and raw databases are often difficult to read as text may appear in random or otherwise non-sequential order. Alternatively the data may be in one order and you may want to display it in another.

A simple sort such as the one in the example above sorts all lines beginning with the first 'token' or blank-delimited string found. On occasion, you may want to sort a table on a column other than the first. **sort** handles this with what is known as a sort 'key' that tells **sort** which column to use. Another common use of **sort** is to display results in reverse order, such as displaying greater quantities first.

One of the first problems users encounter when using **sort** is that results may differ from what was expected. Frequently, lines that should appear in a certain order are displayed out of order. Closer examination usually reveals that the errant lines contain discrepancies – there may be leading blanks or variations in case that affect the order of the sort. The **sort** command has options (flags) for handling these situations that enable you to get the desired results.

FLAGS FOR THE SORT COMMAND

The following flags modify the behaviour of the **sort** command.

Disregarding leading blank spaces (-b flag)

The **-b** flag tells **sort** to disregard leading blanks. For example, suppose your input file, *this.text*, contains the following lines:

```
buffalo
 antelope
  dog
   camel
```

Some entries are preceded with one or more blank spaces; if you were just to **sort this.text**, the results would be:

```
   camel
  dog
 antelope
buffalo
```

This is because blanks are interpreted as any other ASCII character

when sorted using the default sort order. To disregard the blanks, enter **sort -b this.text**, which results in:

```
 antelope
buffalo
   camel
  dog
```

All leading blanks are ignored and the remaining text is sorted alphabetically.

Sorting in dictionary order (-d flag)

The **-d** flag sorts in dictionary order using letters, numbers, and blank spaces. This is useful if you have character strings containing punctuation symbols, such as quotation marks, parentheses, etc, that you wish to ignore.

For example, suppose your input file *this.text* is as follows:

```
Cooper
 (Adams)
'Davis'
"Baker"
```

sort this.text would result in the following:

```
"Baker"
'Davis'
 (Adams)
Cooper
```

To disregard punctuation characters, enter **sort -d this.text** which results in:

```
 (Adams)
"Baker"
Cooper
'Davis'
```

In this case, the symbols are ignored and the remaining text is sorted alphabetically.

Sorting using uppercase only (-f flag)

The **-f flag** converts text to uppercase before sorting. For example, if *this.text* contains the following lines:

```
chess
Bowling
Darts
archery
```

sort this.text would result in the following:

```
Bowling
Darts
archery
chess
```

This is because uppercase letters precede lowercase ones in an ASCII sort. To sort the lines ignoring case, enter **sort -f this.text** which displays:

```
archery
Bowling
chess
Darts
```

In this case, the text is sorted alphabetically without regard to the case of the input strings. Note that the conversion to uppercase is for comparison purposes only – the resulting text is in its original case when displayed.

Performing a numeric sort (-n flag)

The **-n flag** performs a numeric sort. In ASCII, numbers are (sensibly) in numeric order. However, depending on how many blanks precede or follow the digits in the text, numbers can fall out of order if a simple sort is used. Also, an ASCII sort is a byte-by-byte operation, and doesn't take into account place-values, so that '20' is considered greater than '100' as '2' is greater than '1'.

Suppose your input file, *this.text*, contains the following lines:

```
5    Tables
1    Desk
50   Pencils
100  Books
40   Note Pads
10   Lamps
```

sort this.text would result in the following:

```
1    Desk
10   Lamps
```

```
100  Books
40   Note Pads
5    Tables
50   Pencils
```

The ASCII sort starts by comparing the first digit in each column without regard to its decimal position. To sort the file in numerical sequence, use **sort -n this.text**, which results in:

```
1    Desk
5    Tables
10   Lamps
40   Note Pads
50   Pencils
100  Books
```

In this case, all blanks are disregarded and the first ‘token’ found in each line is considered a numeral.

When the **-n flag** is used, the minus sign and decimal point have their usual mathematical meanings, as illustrated in the following example. Suppose *this.text* contains the following lines:

```
1000
100
10
1
0.1
.01
0.001
.0001
-1
-10
-100
-1000
```

Note that the file is currently sorted numerically from the largest positive to the smallest negative number. If you simply entered **sort this.text**, the results would be as follows:

```
-1
-10
-100
-1000
.0001
.01
0.001
0.1
1
```

```
10
100
1000
```

The ASCII sort order position the entries first with those that begin with the minus, followed by those that begin with the period, then the numerals. To sort the file into the proper numerical sequence, enter **sort -n this.text** which would display:

```
-1000
-100
-10
-1
.0001
0.001
.01
0.1
1
10
100
1000
```

The numbers now appear in their proper sequence, taking into account negative numbers and decimals.

Reversing the sort order results (-r flag)

The **-r** flag reverses the order of the sort results. For example, suppose your input file, *this.text*, contains the following lines:

```
  1  airplane
  1  jet
 55  motorcycles
  3  bicycles
120  trucks
  5  boats
230  automobiles
 75  vans
```

sort this.text would result in the following:

```
  1  airplane
  1  jet
  3  bicycles
  5  boats
 55  motorcycles
 75  vans
120  trucks
230  automobiles
```

The default ASCII sort order also happens to be the numeric order as the columns of numbers line up properly. Suppose you wanted to sort this list with the most common vehicles first and the rest in descending order. To do this, use **sort -r this.text**, which displays:

```
230  automobiles
120  trucks
 75  vans
 55  motorcycles
  5  boats
  3  bicycles
  1  jet
  1  airplane
```

In this case, the command performs the sort operation on the file, then reverses the order prior to display.

Checking the results (-c flag)

The **-c** flag checks a file to ensure it is sorted correctly according to the other flags specified on the command line, returning a code with the result of the check. When this flag is used, the command doesn't display the text of a sorted file, as with other flags. Rather, it displays one of two things: either the first line that's out of order or just the command prompt, if the file is sorted correctly.

Suppose your input file *this.text* contains the following lines:

```
avocado
butter
Cheese
doughnut
```

Note that 'Cheese' begins with an uppercase 'C'. If you were to enter **sort -c this.text**, the result would be:

```
sort: disorder: Cheese
```

as an ordinary ASCII sort puts an uppercase 'C' before all lowercase letters.

By contrast, **sort -cf this.text** just returns the command prompt, indicating that the file is correctly sorted (the **-f** flag tells **sort** to ignore case). As *this.text* satisfies this requirement, *sort -cf* returns 'RC=0'.

A **-c** flag is normally associated with **sort** being called from a script,

which then acts according to the return code (a non-zero value indicates that the file is not sorted correctly). An example of its use is a script that checks files prior to a merge operation.

Ignoring non-printable characters (-i flag)

The **-i** flag ignores non-printable characters when sorting. For instance, some text editors insert control characters, such as *Control+M* ('End-Of-Line') and *Control+Z* ('End-Of-File'), in the file, while FTP also inserts non-printable characters if it wrongly makes a binary transfer of an ASCII file from an operating system other than Unix. Your file would then contain the non-printable characters listed above.

Sort -i disregards these characters when sorting.

Specifying a sort key (-k flag)

The **-k** flag specifies a sort key. The key definition is specified as a character string following the flag, as in:

```
sort -k KeyDef filename
```

Sort keys are discussed in detail later in the second part of this article, which appears in next month's issue.

Merging multiple input files (-m flag)

The **-m** flag merges multiple input files. For the merge to be useful, the input files must already be sorted. For example, suppose you had the following three files:

REFGUIDE.WRITERS

```
Mark Baker
Jim Smith
Bill Williams
```

ADMGUIDE.WRITERS

```
Wilma Daniels
Mark Mathers
Sandy Southland
```

USRGUIDE.WRITERS

```
Doug Danford, Jr
Cindy Danwood
William Davis
```

Note that each file has already been sorted alphabetically on the second field. To merge the three files using the **sort** command, enter:

```
sort -m +1 *.writers > all.writers
```

When completed, the file *all.writers* contains:

```
Mark Baker
Doug Danford, Jr
Wilma Daniels
Cindy Danwood
William Davis
Mark Mathers
Jim Smith
Sandy Southland
Bill Williams
```

Note that all lines in the file are sorted on the second field. The **-m** flag tells **sort** to merge the already sorted input files into one sorted output file. The **+1** key tells **sort** to skip the first field and sort on the second and any remaining fields. This spares **sort** some processing, which can be an efficient way to merge large sorted files.

Note that you can use the **-c** flag to determine whether input files are sorted before using the **-m** flag.

Specifying an output file (-o flag)

The **-o** flag specifies an output file for results of the sort. If you enter:

```
sort -o that.text this.text
```

sort sorts the contents of *this.text* and store the results in *that.text*.

A common use of this flag is to redirect the output of the sort back into the file being sorted. This replaces the input file with the data sorted according to the flags on the **sort** command. The **-o** flag allows the output filename to be the same as the input filename, unlike the standard redirection ‘greater than’ sign (>), which requires you to specify different files.

If you enter:

```
sort -o this.text this.text
```

sort sorts the contents of *this.text* and stores them back in *this.text*, overwriting the previous text.

Specifying a field separator (-t flag)

The **-t** flag specifies a field separator character, which is a string following the flag, as in:

```
sort -t Char filename
```

For example, suppose your input file *this.text* contains the following lines:

```
0001:maxwell:032998
0004:martin:043098
0005:cooper:051598
0002:smith:052298
0006:jones:110898
0003:bakerfeld:120998
```

Note that the colon (:) separates the various fields on each line – in this case, a sequence number, a name, and a date field. The file is currently sorted on the date field. If you were to simply enter **sort this.text**, the results would be:

```
0001:maxwell:032998
0002:smith:052298
0003:bakerfeld:120998
0004:martin:043098
0005:cooper:051598
0006:jones:110898
```

In this case **sort** just arranges lines alphabetically, starting with the first character. Suppose you want to sort on the second field (the name field). To do this, enter **sort -t: +1 this.text** which results in the following:

```
0003:bakerfeld:120998
0005:cooper:051598
0006:jones:110898
0004:martin:043098
0001:maxwell:032998
0002:smith:052298
```

The **-t:** tells **sort** that the colon is the field separator, and **+1** tells it to skip one field (the sequence number) to get to the first field on which to sort. The remainder of the text (the name and date fields) is then sorted alphabetically. The **-t** flag is useful for database entries that use separator characters other than blanks, which is often the case with AIX system databases.

Disregarding all but one duplicate entry (-u Flag)

The **-u** ('unique') flag disregards all but one of any duplicate entries. If the data you want to sort contains duplicate entries, and you're not interested in duplicates, use the **-u** flag. For example, suppose *this.text* contains the following:

```
drum
accordion
drum
drum
cello
bass
accordion
cello
drum
```

If you were to enter **sort this.text**, the results would be:

```
accordion
accordion
bass
cello
cello
drum
drum
drum
drum
```

By contrast, **sort -u this.text** would display:

```
accordion
bass
cello
drum
```

Specifying a directory to use for temporary files (-T flag)

The **-T** flag specifies the directory to use for temporary files, as in:

```
sort -T DirName filename
```

Flag	Description
-A	Sorts in ASCII order
-b	Disregards leading blank spaces
-c	Perform a check on the sort results and sets a return code
-d	Sorts in dictionary order
-f	Folds letters to uppercase
-i	Ignores non printing characters
-k Key Def	Specifies a sort key
-m	Merges multiple input files
-n	Performs a numeric sort
-o OutFile	Specifies an output file
-r	Reverses the order of the sort results
-t Char	Specifies a field separator character
-u	Disregards all but one of duplicate entries
-T Dir	Specifies a directory for temporary files

*Figure 1: Flags for the **sort** command*

The **sort** command normally uses the first available directory from the following list to store temporary files during processing:

/var/tmp, */usr/tmp*, */tmp*.

Use the **-T** flag if you want to use a directory other than one of the defaults or you want to use a specific default directory. This could be a benefit if large files are being sorted and space is limited in the default filesystems.

Sorting in ASCII order (-A)

The **-A** flag sorts text in ASCII order. This is the default sort order for the English language and C. If you are working in a locale that uses a different sort order, use the **-A** flag will sort in ASCII order instead. Figure 1 shows the flags for the **sort** command.

David Chakmakian (USA)

From heavy processes to light threads

A program is often required to react to external events that arise in an unregulated way. Such events usually cannot be handled sequentially – only parallel operation allows the program to react appropriately to them.

PROCESSES AND FORK()

Processes are started and directly controlled by the operating system. Processes are kernel-level entities that include such attributes as a virtual memory map, file descriptors, system variables (such as a user-ID), and much more. Each process has its own collection of these attributes. The only way a process can access, query, or change the state of data in this structure is by means of a system call. All parts of the process structure are in kernel space. A user program cannot directly touch any of this data. By contrast, all user code (functions, procedures, etc) and user data (constants, variables, etc) can be accessed directly.

A process can fork itself using the *fork()* subroutine, which creates a copy of the process. It can still detect external influences and react appropriately to them. The new process created by *fork()* is called a 'child process'. Both the child and parent continue execution, starting with the instruction that follows the call to *fork()*. The child is a copy of the parent, so it gets a copy of the parent's data space, heap, and stack. Consider the following example and notice what happens to the data stored in the integer variable *i*. (Note that the continuation character, '➤', is used to indicate that a single line of code maps to several lines of print.)

EXAMPLE OF FORK()

```
#define     _ALL_SOURCE

#include    <stdio.h>
#include    <sys/types.h>
```

```
int
main (void)
{
    int     i;
    pid_t   pid;

    i = 123;
    (void) printf("before fork() : getpid() = %5d, i = %d\n",
                  getpid(), i);

    if ( (pid = fork()) < 0 ) {
        (void) fprintf(stderr, "Error while forking\n");
        exit(1);
    }

    if ( pid == 0 ) {               /* child process                */
        (void) printf("child         : pid = %5d, getpid() = %d,
        ➤  i = %d\n", pid, getpid(), i);
    } else {                        /* parent process               */
        (void) printf("parent        : pid = %5d, getpid() = %d,
        ➤  i = %d\n", pid, getpid(), i);
    }

    (void) sleep(1);
    (void) printf("Who am I?     getpid() = %5d\n", getpid());

    return(0);
}
```

The output of this code is shown below:

```
before fork() : getpid() = 22628, i = 123
child         : pid =     0, getpid() = 19814, i = 123
parent        : pid = 19814, getpid() = 22628, i = 123
Who am I?     getpid() = 19814
Who am I?     getpid() = 22628
```

Notice that the parent receives the child's process ID and also that the child inherits the parent's variables.

THREADS

A thread is a user-level entity. The thread structure exists in user space and can be accessed directly via a thread library call, which is a normal, user-level function. Each thread has its own set of registers (stack pointer, program counter, etc) as well as its own stack. However, the code that the thread is executing is not a part of the thread. The

actual code (functions, routines, etc) is global and can be executed by any thread.

The essential point about threads is that they are user-level entities. Virtually everything that you do to a thread happens at the user level and no system calls are involved. As no system calls are involved, threads are fast. No kernel structure is affected by the existence of threads and no kernel resources are consumed. Threads are cheap. The kernel doesn't even know about the existence of threads.

REENTRANT FUNCTIONS AND THREAD-SAFE LIBRARIES

All threads belonging to a process share the process's state. They all reside in the same memory space and have access to the same functions and data. When a thread alters one of a process's variables, all other threads see this change when they next access the variable. For example, if one thread opens a file to read it, all the other threads can also read the file.

While this provides some very useful benefits, it can also lead to problems. To avoid possible data corruption when two or more threads call the same function at the same time, functions must be 're-entrant'. And to avoid data corruption when two or more functions access the same resource at the same time, functions must be 'thread-safe'. That means that shared resources must be protected by a locking mechanism.

A library is thread-safe when multiple threads can run a library function without data corruption. For this to be the case, all functions in the library must be both re-entrant and thread-safe. In the existing C library, *most* standard functions are re-entrant, but *some* are not. Re-entrant functions and thread-safe libraries are characterized by the suffix *_r*.

WHY USE THREADS?

The main benefits of writing multithreaded programs are:

- *Performance gains from multiprocessing (parallelism)*
  Computers with more than one processor provide multiple

simultaneous paths of execution. Multiple threads provide an efficient method for application developers to use the parallelism of the hardware. Different threads can run on different processors simultaneously with no additional effort on the part of the programmer.

- *Increased application throughput*
  When a process requests a service from the operating system, it must wait for that service to complete, often leaving the CPU idle. Even on a uniprocessor, multithreading allows a process to overlap computation despite the presence of one or more blocked system calls. While the requesting thread must wait, other threads in the process can continue execution.

- *Increased application responsiveness*
  Blocking one part of a process need not block the whole process. While single-threaded processes are unresponsive when performing intensive processing or system work, multithreaded applications remain responsive. Intensive operations can be carried out by independent threads allowing the application to remain active and allow user input. Single-threaded applications are more prone to being swapped out of real memory. This additional work, with its accompanying context switching, results in slow, unresponsive applications.

- *Enhanced process-to-process communication*
  While a multi-processes application uses traditional IPC (Inter Process Communication), a multithreaded application can simply use the process's shared memory.

- *Efficient use of system resources*
  The operating system must allocate a complete set of resources for process structure. By contrast, an application can create hundreds or even thousands of threads with only a minor impact on the system's resources. Threads use a fraction of the resources used by a process.

- *Ability to use the inherent 'threadedness' of distributed objects*
  Distributed objects and request brokers are inherently multithreaded, even when used by a single-threaded application.

- *One binary for uniprocessors and multiprocessors*
  In most older parallel processing schemes, it was necessary to tailor a program for the individual hardware configuration. With threads this customization is not required as the multithreaded paradigm works well irrespective of the number of CPUs. A program can be compiled once, and it will run acceptably on either a uniprocessor or a multiprocessor (on which) it will run faster.
- *Ability to create well-structured programs*
  A traditional program that tries to do too many things at once is quickly swamped with lots of complex code required to coordinate these tasks. A threaded program can do the same tasks with much less, and far simpler, code.
- *There can be a single source for multiple platforms*
  Many programs must run on numerous platforms. With the POSIX threads standard, it is possible to write a single source and recompile it for different platforms.

POSIX THREADS

In June 1995 the IEEE ratified the POSIX.1c standard for threads. These threads, often called 'pthreads', are defined in a thread library containing over sixty functions. The standard does not define whether these threads are implemented at the user-level or kernel-level. This results in differences in scheduling, and depends on how the operating system handles threads.

AIX Version 4, which itself has a multithreaded kernel, supports the draft 7 version of the POSIX specification. The POSIX.1c draft 10 is to be the final standard, and it's IBM's intention to comply with it in due course. The programs that follow were tested using AIX 4.2.1.

CREATING POSIX THREADS

A thread is created using the function *pthread_create()*. The first parameter is the return value of the new thread's identifier. The second attribute is a pointer to an attribute object describing the thread and containing values such as stack size, stack address, detach status,

scheduling policies, priority, and the like. A null pointer means that default values are to be used. The third parameter is a pointer to the function that is to be executed as the thread's starting function. The fourth and last parameter points to the starting function's parameters.

EXAMPLE OF CREATING THREADS

```
#define     _ALL_SOURCE

#define     NUMBER_OF_THREADS   10

#include    <pthread.h>
#include    <stdio.h>
#include    <unistd.h>

void *
handle_client(void *parameters)
{
    (void) fprintf(stderr, "Thread %s is doing its work...\n",
    ➤  parameters);
    sleep( atoi(parameters) );  /* simulate work */
    (void) fprintf(stderr, "Thread %s is finished!\n", parameters);

    return NULL;
}

int main(void)
{
    int         i, ierror;
    char        sbuffer[NUMBER_OF_THREADS][4];
    pthread_t   Thread[NUMBER_OF_THREADS];

    for ( i = 0; i < NUMBER_OF_THREADS; i++ ) {
        (void) sprintf(sbuffer[i], "%d", i);
        if ( (ierror = pthread_create(&Thread[i], NULL, handle_client,
                                      (void *) sbuffer[i] ) ) != 0 ) {
            (void) fprintf(stderr, "pthread_create() error %d\n",
                           ierror);
            exit(1);
        }
        pthread_detach(Thread[i]);
    }

    /* wait until all threads are surely finished */
    sleep((NUMBER_OF_THREADS * (NUMBER_OF_THREADS / 2)) + 5);

    return(0);
}
```

Compile the above example using the following command:

```
xlc_r -o example1 example1.c -lpthreads
```

In case you don't have the **xlc_r** compiler, there's no need to worry – make it yourself by linking the existing **xlc** C compiler to it:

```
ln /usr/bin/xlc /usr/bin/xlc_r
```

If you take a look at the */etc/xlc.cfg* configuration file you'll see that **xlc_r** has similar, yet slightly different, entries when compared with **xlc**. This is because it uses thread-safe, re-entrant libraries.

A thread terminates when it returns from its starting function. This can occur with a *return()* statement or by using the thread library's *pthread_exit()* function. The thread's exit value is either its return value or *pthread_exit()*'s argument value, depending on how it terminates. The thread also terminates when its creator dies. A word of warning: do not terminate threads using *exit()*, as this terminates not only the thread but also the entire process. The call to *sleep(NUMBER_OF_THREADS + 5)* in the program provides enough time for the threads to be created and exit by themselves.

pthread_join() is used by a thread to wait for another thread to exit, while *pthread_detach()* indicates that you have no interest in the thread's return value.

SYNCHRONIZING POSIX THREADS

As multithreading brings with it the risk that a resource may be over-used, POSIX threads support 'mutex' locks ('mutually exclusive'). Notice the use of the functions *pthread_mutex_init()*, *pthread_mutex_lock()*, and *pthread_mutex_unlock()* in the modified version of the previous example shown below.

EXAMPLE OF SYNCHRONIZING THREADS

```
#define     _ALL_SOURCE

#define     NUMBER_OF_THREADS   10

#include    <pthread.h>
#include    <stdio.h>
```

```
#include    <unistd.h>

pthread_mutex_t db_mutex;

void *
handle_client(void *parameters)
{
    pthread_mutex_lock(&db_mutex);
    (void) fprintf(stderr, "Thread %s is doing its work...\n",
                   parameters);
    sleep( atoi(parameters) );  /* simulate work */
    (void) fprintf(stderr, "Thread %s is finished!\n", parameters);
    pthread_mutex_unlock(&db_mutex);

    return NULL;
}

int main(void)
{
    int         i, ierror;
    char        sbuffer[NUMBER_OF_THREADS][4];
    pthread_t   Thread[NUMBER_OF_THREADS];

    pthread_mutex_init(&db_mutex, NULL);

    for ( i = 0; i < NUMBER_OF_THREADS; i++ ) {
        (void) sprintf(sbuffer[i], "%d", i);
        if ( (ierror = pthread_create(&Thread[i], NULL, handle_client,
                                      (void *) sbuffer[i] ) ) != 0 ) {
            (void) fprintf(stderr, "pthread_create() error %d\n",
                           ierror);
            exit(1);
        }
        pthread_detach(Thread[i]);
    }

    /* wait until all threads are surely finished */
    sleep(NUMBER_OF_THREADS + 5);

    return(0);
}
```

ADDITIONAL PTHREAD FUNCTIONS

pthread_cond_init() initializes a condition variable. A thread then waits with *pthread_cond_wait()* until the condition variable is triggered by the right value. It can then set a mutex lock to use an unshareable system resource. If more than one thread is waiting for the condition

variable, the operating system's scheduling policies determine which thread is scheduled.

Some of the sixty other *pthread* library functions deal with the scheduling of threads, their signal handling, thread specific data, and so on.

SUMMARY

While the example in this article may seem rather trivial, it nevertheless demonstrates the capabilities of a multithreaded program compared to traditional, process-generating programs. Thread programming in the AIX environment is still relatively new and is often not specifically supported. Nevertheless, its power and flexibility mean that it warrants a developer's attention.

Werner Klauser
Klauser Informatik (Switzerland)

New RS/6000 hardware

Traditionally autumn is the time of IBM's RS/6000 and AIX announcements. This season's crop is particularly varied: it contains announcements of low and high-end RS/6000 systems and of major enhancements to the AIX operating system.

RS/6000 ENTERPRISE SERVER S70 ADVANCED

The new RS/6000 S70 Advanced (S7A) is based on the existing Model S70, with enhancements in the areas of performance, capacity, and RAS ('Reliability, Availability, and Serviceability'). The Model S70 can be upgraded with all the new features of S7A, which is very important to protect investments made by existing users.

The system can be configured with up to three four-way CPU boards equipped with 262 MHz RS64-IIs. Each processor comes with 1 GB of L1 cache and 8 GB of L2 cache. The cache size is double that of the previous generation of 125 MHz RS64 CPUs available with the Model S70. The system comes with a 9.1 GB UltraSCSI hard disk and 1 GB of RAM. (The S70 comes with a 4.5 GB SCSI-2 F/W hard disk and 512 MB of RAM.) The basic S7A comes with two UltraSCSI adapters, compared with 2 SCSI-2 F/W adapters in the S70. The new system supports up to 32 GB of RAM when 2 GB memory cards are used.

System reliability has been enhanced by the introduction of the new 'Advanced I/O' drawer. This drawer, which is three EIA units higher than an S70 I/O drawer (which is seven EIA units tall) has dual AC power supplies and N+1 cooling fans. The drawer has additional benefits, such as SSA disk support in secondary I/O drawers, ultra SCSI support, separate cabling for dual six-packs for improved reliability and performance, and support for the installation of PCI cards without the need to remove drawers.

SP SYSTEM ATTACHMENT ADAPTER

The SP System Attachment Adapter allows the integration of models S70 and S7A in the framework of an RS/6000 SP system. The adapter allows the an S7X SMP server to be connected to the switch of an SP system. The S7*x* system effectively becomes an SP node, and is managed from the SP's Control Workstation under the IBM Parallel Systems Support Program for AIX. This integration enables the utilization of the S7X's extended memory and OLTP performance for complex data processing applications, such as ERP and data mining.

The adapter must be installed in slot 9 of the primary I/O drawer of the S7*x* server. Additionally the machine must have at least one 10 Mbps Ethernet adapter. The adapter is supported only under AIX 4.3.2 and above.

ADVANCED POWER3 MICROPROCESSOR

The POWER3 processor, along with its integration in the RS/6000

43P Model 260 workstation/server, is perhaps the most significant feature of the whole announcement. This 64-bit microprocessor, based on the PowerPC architecture, provides the floating point performance needed to support demanding scientific and engineering applications. Being the first of the new generation of chips (which are to be the basis of future computationally-intensive SMP servers), the chip runs at only 200 MHz. In spite of this, it has clocked 13.2 SPECint95 and 30.1 SPECfp95 and is the most powerful floating point chip available on the market today.

The microprocessor has eight execution units and is able to perform many operations concurrently. Up to four instructions can be dispatched at a time and executed out of order, while still assuring program integrity. The chip has a 32 KB instruction cache and 64 KB data cache. To ensure a sufficiently high memory transfer rate the chip provides 6.4 GB/s bandwidth to cache (load/store L1 from/to L2) and 1.6 GB/s to memory (load/store L1 from/to memory).

THE IBM RS/6000 43P MODEL 260

The POWER3 microprocessor is the engine behind the new 43P Model 260. This 64-bit Symmetric Multi-Processing (SMP) system is designed to provide extended floating-point capabilities. The Model 260 can be configured with up to two 200 MHz processors with 4 MB of Level 2 (L2) cache per processor. The system memory can be expanded up to 4 GB. You should note, however, that 4 GB of RAM requires AIX version 4.3.2 – version 4.2.1 supports only up to 3 GB.

A Fast Ethernet adapter, UltraSCSI adapter, and service processor are integrated within the machine. Other connectors available as standard are: tablet, keyboard, mouse, two serial, one parallel, and stereo audio. The system provides two disk drive bays, one of which is used by the disk supplied with the system (either 4.5 GB or 9.1 GB). Additional disks can be installed in the available bay. The system is expandable with up to 27.3 GB of internal disk storage when three 9.1 GB disks are installed. Two of the media bays are occupied by a 1.44 MB 3.5" diskette drive and a 32x max CD-ROM. Five PCI slots are available (two 64-bit, 50 MHz slots and three 32-bit, 33 MHz slots). All are long slots. The system’s performance looks particularly impressive when

compared with the previous generation of SMP workstations Model 240, as shown in Figure 1.

Benchmark	Model 240	Model 260	Improvement
SPECint_base95	7.8	12.5	60%
SPECfp_base95	5.6	27.6	393%
SPECint95	8.71	13.2	52%
SPECfp95	5.87	30.1	413%
SPECint_base_rate95			
1 processor	70	111	58%
2 processors	132	222	68%
SPECfp_base_rate95			
1 processor	50.1	243	385%
2 processors	89.5	468	423%
Relative OLTP (1/2 procs)			
1 processor	3.7	11	184%
2 processors	5.2	21	304%

Figure 1: Comparing new models with the Model 240

THE IBM RS/6000 43P MODEL 150

The 43P Model 150 is the successor to the 43P Model 140 workstation, and is designed as an entry-level workstation or workgroup server. It uses a 375 MHz PowerPC 604e processor with improved memory controller and SDRAM memory, and has a memory bus speed of 83 MHz (compared with 66 MHz for the Model 140). System memory can be expanded up to 1 GB. A Fast Ethernet adapter and UltraSCSI adapter are integrated within the machine. It comes with the same standard connectors as the Model 260, and, like the Model 260, provides two disk drive bays, one of which is occupied by the system disk (either 4.5 GB or 9.1 GB). Additional disk can be installed in the available bay. The system is expandable with up to 27.3 GB of internal disk storage when three 9.1 GB disks are installed. Two of the media bays are taken up by a 1.44 MB 3.5" diskette drive and a 32x max CD-ROM. Five 32-bit PCI slots are available (two short and three long).

With the exception of floating point performance, the performance of the system shows an improvement proportional to the increase in processor speed relative to fastest Model 140 running at 332 MHz.

Benchmark	Model 140	Model 150	Improvement
SPECint_base95	12.5	14.5	16%
SPECfp_base95	5.99	9.85	64%
SPECint95	12.9	15.1	17%
SPECfp95	6.21	10.1	63%
Relative OLTP	5.3	6.0	13%

Figure 2: Model 150 compared with the Model 140

POWER GXT3000P GRAPHICS ACCELERATOR

The POWER GXT3000P graphics accelerator is IBM's top-of-the-range graphic card. Based on industry standard benchmarks, its performance is as much as five times better than that produced so far by IBM's workstations (Figure 3).

Model	43P Model 260	43P Model 150	43P Model140 (332 MHz)
uni/multi-processor	uni/multi-processor	unipro-cessor	unipro-cessor
Graphical card	GXT3000P	GXT3000P	GXT800P(T)
PLBwire93	436.9/627.4	257.3	157.4
PLBsurf 93	610.8/866.2	468.9	264.9
CDRS-03	218.17	94.76	40.97
GLperf line	7.05 M	4.9 M	1.5 M
GLperf triangle	5.19 M	3.7 M	924 K

Figure 3: GXT3000 compared with fastest previous model

The card, which is supported only on the 43P Model 150 and 43P Model 260, fits both 32 and 64-bit PCI slots, taking up two slots. The list of characteristics of the accelerator is very impressive:

- 1280 by 1024 maximum resolution at up to 85 Hz refresh rate (1024 by 768 resolution at 120 Hz)
- 8-bit and 24-bit double-buffered colour buffers
- 24-bit Z-buffer
- 8-bit overlay buffer
- 8-bit window ID buffer
- 8-bit stencil buffer
- 8-bit double-buffered alpha buffer
- Eight 256-entry hardware colour tables
- Separate gamma correction table
- Hardware accelerated lighting
- Increased sub-pixel addressing (compared with previously available adapters)
- Support for ‘face culling’
- Support for OpenGL polygon offset
- Support for OpenGL polygon mode
- Hardware support for OpenGL specular lighting
- 32 MB texture memory
- Trilinear texture and 3D texture mapping
- Video Support
- Point sampling and bilinear scaling
- Colour space conversion
- Stereo sound

- 12-bit RGB on a per-window basis
- Preserved aspect ratio.

STEALTH BLACK MONITORS

IBM has introduced a new line of monitors to match its new advanced workstations: Models P72, P92, and P202. These monitors are compliant with the latest environmental and power consumption regulations. Models P202 and P92 have two separate video inputs that can be connected to AIX workstations and PCs simultaneously (the active input is selected by means of buttons located on the front panel). Models P202 and P92 support resolution of up to 1600 by 1200, while Model P72 supports a maximum resolution of 1280 by 1024 pixels.

GIGABIT FIBRE CHANNEL ADAPTER

This adapter is the first implementation of the Fibre Channel Arbitrated Loop (FC-AL) storage interconnect architecture on the RS/6000. FC-AL provides a high-speed interconnect running at 100 MB/s for a single loop of storage devices connected to a card. The technology is based on an arbitrated loop that allows only one at a time device to perform I/O on a bus. The card supports the following two types of cable:

- Multimode 550/125 micron fibre with a minimum length of two metres and a maximum of 500, using SC connectors and providing a throughput of 1,062.5 Mbps.
- Multimode 62.5/125 micron fibre with a minimum length of two metres and a maximum of 175, using SC connectors and providing a throughput of 1,062.5 Mbps.

Only one initiator (card) is supported on the loop. The card has been announced for Models S70/S7A under AIX 4.3 (or later). Support for the F50, H50, and SP is expected to be announced in 1999.

GIGABIT ETHERNET ADAPTER

Gigabit Ethernet is an extension to the 10/100 Mbps IEEE 803.2 Ethernet standard, offering a higher bandwidth of 1,000 Mbps. The

card provides one full-duplex 100Base-SX connection. The adapter has the following characteristics:

- Full duplex performance
- Support for 'Jumbo Frame Rates' (9 KB) under AIX 4.3.2 or later
- Hardware support for TCP, UDP, and IP checksums (needs AIX 4.3.2 or later)
- PCI Rev 2.0 and 2.1 compliant
- PCI bus mastering and 64-bit data and address buses running at 33 MHz or 66 MHz (requires 64-bit slot)
- Supported on Models 260, F50, H50, S70, and S7A
- Supported under AIX 4.2.1 and AIX 4.3.2
- Requires 850 nm multimode 62.5 or 50 micron fibre cables of up to 500 m.

ARTIC960RXD DIGITAL TRUNK PCI ADAPTER

This intelligent communication adapter provides voice processing for up to four T1 or E1 digital trunks lines, providing connectivity for up to 96 T1 or 120 E1 voice channels in a single PCI slot. The card is used by IBM's DirectTalk Software. It's supported on models 140, F50, and H50 under AIX 4.2.1 and AIX 4.3.2.

ARTIC960HX 4-PORT PCI ADAPTER

This card has been announced previously as a replacement for the four-port ISA multiprotocol communications controller adapter, which is now withdrawn. The card provides four ports, which can be RS-449, X.21, X.25, or X.35 running at up to 2.0 Mbps. The hardware provides support for X.25, SNA, and BiSync WAN protocols. The card is supported under AIX 4.2.1 and AIX 4.3.2.

4 GB EXTERNAL SLR5 QIC TAPE DRIVE

This external drive supports quarter-inch tape cartridges. It features the improved IC-4GB format, while retaining compatibility with the

QIC-120/150, QIC-525, QIC-1000, and QIC-2GB formats. Up to 4 GB of data can be stored on a single cartridge (8 GB with compression), which is a 300% improvement over the previous model. Sustained data transfer rate is 380 Kbps without compression or 760 Kbps with.

32X SCSI CD-ROM DRIVE

The main feature of this internal CD-ROM drive, which is to be installed in all new RS/6000 models, is an improved transfer rate of 4,800 Kbps. It is to be supported under all versions of AIX.

REFERENCES

1 *RS/6000 Scientific and Technical Computing: POWER3 Introduction and Tuning Guide*, SG24-5155, IBM Corporation

2 *POWER3: Next Generation of 64-bit PowerPC Processor Design*, IBM white paper, 1998

3 *The GXT3000P Graphics Accelerator*, IBM white paper, 1998

4 *The IBM RS/60000 43P Model 260*, IBM white paper, 1998

5 *RS/6000 Model S70 Family – Technology and Architecture*, IBM white paper, 1998

6 *RS/6000 Model S70 RAS*, IBM white paper, 1998.

System Engineer (Israel)

If you'd like to submit an article to *AIX Update*, please send it to the editor, Harold Lewis, at *harryl@xephon.com*. Articles may be submitted in ASCII text or in one of a number of wordprocessor formats – for guidelines on creating and submitting articles to *AIX Update*, please refer to our *Notes for contributors*, which may be downloaded from Xephon's Web site (*http://www.xephon.com*).

Performance reports

This month's instalment concludes this article on performance reports, the first part of which appeared in last month's issue.

THE EXCEL SET-UP

In order to run the application successfully, it's essential to set up Excel as follows:

1 Maximize Excel's window.

2 Ensure the 'Full screen' option in the *View* menu is off.

3 Set the screen resolution to 800 by 600 pixels (this is set in the *Display Control Panel*'s *Settings* tab).

4 Excel should be set up to display the *Formula Bar* and *Status Bar* (both these options are on the *View* menu).

There should be three rows of toolbars below the Excel menus. The reason why the display needs to be set up this carefully is that the application uses screen position when creating charts.

Your screen in Excel should look similar to the one in Figure 1. Note that it doesn't matter exactly which toolbars are used as long as there are three below the menu bar.

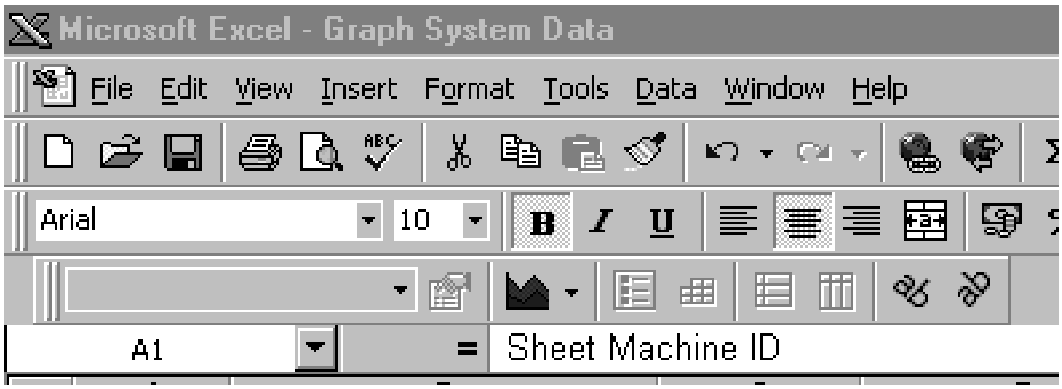

Figure 1: The application's interface in Excel

WINDOWS EXPLORER SETTINGS

Windows Explorer needs the following settings to allow the application to open files correctly.

1 Start *Windows Explorer*.

2 Go to the *View* menu.

3 Select the *Option* menu item.

4 Click the *View* tab.

5 Ensure that *Display the full MS-DOS path in the title bar* is not selected.

6 Select the option *Hide MS-DOS file extension for file types that are registered*.

COLLECTION INFORMATION

Before setting up the application, it's worth collecting the following four important pieces of information about your set-up:

1 The path of the directory where you intend to keep report workbooks. The application will save all reports produced in this directory.

2 The full name of the RS/6000 server on which you wish to report. This is used, where necessary, to distinguish the various systems, and appear on reports.

3 The path of the directory where data files from the collection scripts are to be kept (they must be accessible from the PC running the application). This is the directory to which data files are transferred using FTP or another suitable file transfer method from RS/6000s.

4 The value of the *MACHINE_ID* variable from the **control_load** collection script. This is used as the file name when the relevant data file is built from within the application. You'll encounter problems if this value is not consistent between scripts.

Note that the value of the *MACHINE_ID* variable from the

format_command_output script is required later when you set up the reports from the text data files. The *MACHINE_ID* from the **control_load** script is used to set up a default report that can be used as the basis of your own reports.

Once the Graph System Data workbook is created and saved with the code that appears in the first part of this article (see *Performance reports*, *AIX Update* Issue 38), two macros can then be run to set up the collection procedure.

To run macros from Excel, open the *Tools* menu and select the *Macro* option followed by the *Macro* sub-option. Select the required macro from the list and click *Run*. This is also where to edit macros, if necessary – just click *Edit* instead of *Run* and Excel will go straight to the macro code.

THE GSD_RUNSETUP MACRO

The **gsd_runsetup** macro should be the first one run after the Excel workbook is created and saved as 'Graph System Data'. This macro sets up all the worksheets needed by the application. It should be run after you collect the information described in the previous section, as the macro requests the directory where the reports are kept. When specifying directories, it's important to include the final backslash ('\') of the path name (eg: '*c:\data*'), otherwise problems occur. Note that the information is requested by a plain input box, and no error checking is done either on the existence or the accessibility of the directory. This is the only piece of information required by this macro. When the macro completes execution, the necessary building blocks for the application are complete. What remains is to build the report profiles. To help in setting up reports, a default report set up macro is included.

THE DEFAULT_REPORT_SETUP MACRO

The **default_report_setup** macro is used to set up a report based on **sar** information (using the **-u** option) obtained via the **control_load** and **format** scripts. This **sar** file contains data on the RS/6000's CPU performance, and is used to set up a default report to which other

information can be added. Three more items of information are required by the macro and are prompted for:

1 The full name of the RS/6000 server you are reporting on.

2 The directory where data files from the collection scripts are kept.

3 The value of the *MACHINE_ID* variable from the **control_load** collection script.

This data is used to generate the worksheets necessary for the three reports available from the application. When the macro completes, the basic information for the first run is complete. The Excel workbook is saved at the end of the macro. If the workbook is closed and then re-opened, the first run will take place. The *Auto_Open* name is used to run a macro called **builder**, which controls report generations. Before a run is attempted, make sure the data text files are transferred to the relevant directories.

When the macro completes, three new worksheets are created. They are as follows, where *BOX* is the full name of the RS/6000:

- *BOX*
- *BOX ave*
- *BOX indy*.

In addition, two empty Excel workbooks are created in the *reports* directory. They are as follows (*BOX* is the full name of the RS/6000):

- *BOX summary* (for daily reports)
- *BOX monthly summary* (for monthly reports).

Before continuing, ensure that these worksheets have indeed been created. If there are any problems, refer to the entries given during the set-up macro procedures.

APPLICATION WORKSHEETS

The application, when set up, will have created several worksheets and dialogue sheets. The following worksheets are vital to the proper running of the application:

- *Control*
- *Control ME*
- *Data*
- *Calculations*
- *Sheet5*

Depending on how many reports are set up, extra worksheets will be created for each additional report.

CONTROL WORKSHEET

This worksheet is used by the daily or weekly reports. It is the key sheet that generates all the daily reports and sets out the scope of these reports. Reports are run every day in the same order as they are set up, the exception being the first day of the month (this enables the Control ME sheet to be activated for month-end reports).

The row containing 'END' in each of the specified columns should be left alone, as it is used to determine the last row of the reports.

Sheet machine ID column

This entry refers to the worksheet in the workbook that contains the charts needed to generate reports for the specified RS/6000. The worksheet must exist and it must have exactly the same name as appears here. This enables you to set up different reports that may be needed by specific groups. For example, an overview may be produced for managers, while a more detailed report is produced for administrators. These can be set up using the same data, though different summary reports are used (see the summary file for further explanation). As a worksheet is created using only this input, it must contain only characters that are valid for a worksheet name (refer to Excel documentation for a list of these characters).

Data directory

This entry points to the directory that contains the text data files created by the collection scripts. The directory must be accessible

from the PC and the path must end with a backslash (for example '*c:\data*' not '*c:\data*') – this is required as only the file name is appended to the path.

Full box name

This is the name of the RS/6000 that's being reported on. The name entered here appears in all reports generated.

Summary File

This is the workbook created to hold the summary reports. It must be in place before the program is run, as the program tries to open it before processing takes place.

Report date from

This is the number of days subtracted from the current date to yield the date on which the report should commence. For example, if the value is '2' and the program is run on June 3, 1998, then the report starts from June 1, 1998. It's important that this figure does not pre-date the start of collections, as it is used to generate the data file names. Only numeric data should ever be entered in this field or unpredictable results may result.

Report date to

This is a number that is subtracted from the current date to yield the date to which the report should extend. For example, if the value '1' is entered in this field and the program is run on June 3, 1998, then the report extends to June 2, 1998. Note that, if the start and end dates of a report are not in the same month, two data files are used in the reporting process. This somewhat slows down the generation of reports. Only numeric data should ever be entered in this field or unpredictable results may result.

'Report time from' and 'Report time to'

Time values in this field should be in the format *HH:MM*; any other format may yield unpredictable results. Specifying this value determines the time at which reports start ('*Report time from*') or end ('*Report time to*').

Load File Extension

This can be either *.TXT* or *A.TXT*.

- *.TXT* refers to daily data from the collection routines
- *A.TXT* refers to averaged data from the collection routines (**sar** data).

Any other entry in this field will not generate a successful report.

Data file type

This will be either *DAT* or *AVE*

- *DAT* refers to the daily data from the collection routines
- *AVE* refers to the averaged data from the collection routines (**sar** data).

Any other entry in this field will not generate a successful report.

CONTROL ME WORKSHEET

This worksheet is used for monthly reports. It controls the reports and is generated the first time the program is run each month. The set up and format of data in each of its columns is exactly the same as that of the *Control* worksheet unless specified below.

The row containing *END* in each of the specified columns should not be altered as it's used to establish the last row of reports.

Summary file

If individual reports are specified, then the entry here should be 'None' (it has to be exactly 'None', without the single quotes, or the program will not run properly).

Report date from

If monthly averaged reports are specified, then the entry in this field (which must be a number) must match the entry in *Report date to*. As the *Control ME* sheet is activated the first time the program is run each month, the value entered here should be large enough to take the report into the previous month. For example, if you run the application only

on Monday to Friday, then the figure in this field must be at least '4' to ensure the previous month's data is processed.

Report date to

If individual reports are specified, then the entry in this field should be 'ALL'. This is the only accepted entry for this type of report. For monthly averaged reports, the figure entered here must match that in the *Report date from* field.

'Report time from' and 'Report time to'

No value needs to be entered in either of these fields for monthly averaged reports.

Load file extension

For monthly averaged reports, the file extension should be '*A.TXT*'; for individual reports, it should be '*.TXT*'.

Data file type

For monthly averaged reports, the file extension should be '*AVE*'; for individual reports it should be '*DAT*'.

SHEET5 WORKSHEET

This sheet is used to create reports for the specified RS/6000s. It must match an entry in the *Sheet Machine ID* column of either the *Control* or *Control ME* worksheets. This is where reports are created. Do not delete this worksheet as it is used to create new worksheets.

File

This specifies the start of the file name to use. For example, if *BOXA* appears here, then the *MACHINE_ID* variable is '*BOX*' and the specified data file is '*A*'.

Number of charts

This is the number of charts that are created from data in the specified file. As more charts are added, the size of the report increases. This can

be a problem on systems that are short on memory. If you find that your system runs out of memory, then reduce the number of charts produced for each report.

Reduce column scale

This contains references to columns that are to be rescaled when the data file is opened. If graphs are based on data that differs greatly in size, this option can be used to reduce columns by orders of magnitude. Hence, if data in column *D* is in the tens, in column *E* in thousands, and in column *F* in ten thousands, the resulting entry to rescale the sets of data is:

```
E100,F1000
```

The entries in this column may contain only a column letter followed by a one and a number of noughts, and entries should be separated by a comma; any other character may produce unpredictable results. It is possible only to reduce columns – there is no option for increasing them.

Add to save name

This option is used to add an extension to individual reports' file names. This is useful when many reports are derived from one data file, as the same name is always used to save the report workbook. For example, disk reports could be grouped by the disks' volume groups, and this could be used as the extension. All reports derived from the *J* file would be saved with the extension *VG*. The entry *rootvg* would yield the filename '*BOXJrootvg*' for the workbook.

Name of chart

This name is used as the title of the chart and the worksheet containing the graphs. As it's used as the name of the worksheet, only valid worksheet name characters may be used.

Columns to chart

This entry specifies the columns in the data file that are used to create charts. It's in the form of a character string that contains the relevant column letters. (The first part of this article, see *AIX Update* Issue 38,

contains two tables with details of the data in each column.) For example, to chart **sar** information on CPU from the *U* daily data file, the entry would be '*CDEFG*', where:

- *C* charts the *time* column
- *D* charts *%sys* CPU column
- *E* charts *%usr* CPU column
- *F* charts *%wio* CPU column
- *G* charts *%idle* CPU column from the data file.

If the data file runs to more than 26 columns, then the format of column 27 on is as follows: '*CDA+A*', where '*A+A*' is column 27. There is a limit of 256 columns for the data file, which is a result of Excel's limit of 256 columns per worksheet.

'X axis title' and 'Y axis title'

These are the titles of the x- and y-axes of the chart respectively.

Type of chart

Five types of chart are supported; they are:

1 *xlLine*

2 *xlLineMarkers*

3 *xlLineStacked*

4 *xlColumnStacked100*

5 *xlColumnStacked*.

Refer to Excel's documentation for an explanation of these charts.

Design type

Two different layouts are available for charts on report pages. The first is six-to-a-page in a 2 by 3 format and the second is three-to-a-page in a 1 by 3 format (only on a landscape page).

Correspondingly, the two possible entries for *Design type* are:

1 ‘2 by 3’ format

2 ‘1 by 3’ format.

This setting is ignored if you are producing monthly reports, in which each chart appears on its own worksheet. Therefore the only reports affected are *Daily* and *Individual* reports.

The format and positioning of charts was developed for a LaserJet 4si printer; changing margin settings in the page layout may be necessary to allow charts to fit your printer’s page output. There is a macro to automate this called **page_setup_margins** – to run it you need to uncomment some statements in the main **builder** macro.

Footer text

Text in this entry appears in each report chart sheet’s footer.

Standard scale

The two options available are ‘Y’ and ‘N’, which control the use of a standard scale. ‘Y’ tells the application to use the same scale on all charts. This requires it to detect the largest number in the data file for use in the standard scale. ‘N’ tells the application that each chart is to have its own scale.

Round up max scale

If a standard scale is used, then this field should conform to one of two options:

- A positive number is used as the *MAXIMUM* value on all charts.
- A negative number tells the application to round up the maximum value in the data file by the number of decimal places specified. For example, if ‘–3’ is entered, and the maximum value in the data file is 950, then the maximum of the scale should be 1,000.

DATA WORKSHEET

Do not delete this worksheet as it is vital to the application. It holds information used for building reports. The information comprises

named ranges used by dialogue sheets to build worksheets. The worksheet also contains the directory where report workbooks are saved (if you need to change this for any reason, do so manually by entering the new value in row 1 column 7). This worksheet also determines which control worksheet is processed (that is, whether *Control* or *Control ME* is used when the Excel workbook opens), which depends on the name of the month in row 1 column 8. All other information on this worksheet is controlled and modified by macros used to run the dialogue sheets, so don't change it manually.

CALCULATIONS WORKSHEET

Do not delete this worksheet as it is vital to the application – it is used to perform the column reduction procedure selected in the report set-up options (this was discussed earlier under the heading *Sheet5 Worksheet*). It holds processed performance data and is used to 'align' heterogeneous data, enabling the data to be presented on one chart. This is used in preference to a secondary scale in the Excel charts.

CHART SETUP

The main function of this application is to generate charts based on RS/6000 performance data, so the design and presentation of these charts is an important feature of the reports. The application produces default charts that meet our reporting requirements. Chart set-ups can be altered by changing the VBA code used to produce them. The two macros that control this are **format_chart** and **graph_data**. The application uses the same standard chart format for all reports. This is done for consistency and results in clearer presentation of the data. These charts are based on default Excel charts to which modifications are made. The only formatting option controlled from worksheets themselves is the *Type of chart* column in *Worksheet5*. If the presentation of the charts does not meet your requirements, then you'll have to alter the VBA code to modify them. Another method is to set up a custom type chart in Excel based on your formatting requirements.

A brief explanation of the **format_chart** and **graph_data** macros mentioned above follows.

FORMAT_CHART MACRO

This macro formats charts to a standard design. You can alter it either directly by modifying the code itself or indirectly by recording your changes in a macro and then attaching the resulting code to the end of the macro. For example, you could click on a chart, then click on the 'record macro' button in the Visual Basic toolbar, and make changes to the chart using the various options available. Then cut the resulting code from the new macro and paste it on the end of the **format_chart** macro. The code you take must lie between the *Sub Macro1()* and *End Sub* lines, otherwise you'll get errors. Remember that this just controls the appearance of the chart – titles are controlled by worksheet entries and can be changed on the relevant worksheet column.

GRAPH_DATA MACRO

This macro sets up the five default charts. To add another default chart, the code needs to be changed. Look for the following code in the macro:

```
If CHART_TYPE = "xlColumnStacked100" Then
ActiveChart.ChartType = xlColumnStacked100
End If
```

New chart types can be added by changing the value of the *CHART_TYPE* variable, which is set by the *Type of chart* entry, in which case VBA code needs to be added so that the active chart is set to the new type. This macro also positions charts on worksheets and formats pages for printing.

USING THE EXCEL APPLICATION

The application has been designed to be flexible and, as such, can take a while to get used to. Using default report creation is a good place to start, as all report types are created. From this base all other reports can be added by following the procedures described here. There is little error checking on the values in worksheets, so it is important to ensure that the entries are correct.

The application is geared towards daily, weekly, or monthly reports. The month flag on the *Data* worksheet controls which report is

generated. Depending on this setting, the different runs take place. For example, if the flag does not match the current date when the application runs, the *Control ME* report is generated. On the other hand, if the date matches the current date, the *Control* report is generated. This doesn't mean you have to have this split – you can add the daily reports to the *Control ME* worksheet, if this is what you require. Similarly, if you need to generate individual or monthly reports daily, these can be added to the *Control* worksheet.

Our set-up uses a 100 MHz Pentium PC with 40 MB of memory running Windows 95 and Office 97, which is adequate for this application in our environment. We currently report on three systems and have no problems producing the three types of report. Excel 97 can be memory-intensive, so, if your system is short on memory, you should build the reports gradually. We use System Agent to run the application overnight, which means that any PC can be used. System Agent is an application that comes with Windows 95's *Plus!* pack; however, any other scheduling program for Windows could be used. As the *Auto_Open* name is used, a shortcut to the Excel workbook can also be used to run the application. Note, however, that the application should be the only one running on the PC, otherwise it will not complete properly.

POTENTIAL PROBLEMS

Extremely slow run times

There is a problem in Office 97: if Outlook 97 is used to journal files that greatly increases the time it takes to close Excel workbooks. This problem was resolved with the service pack for Office 97. If you do not want to install this service pack or don't have access to it, turn off the journalling in Outlook 97. Start Outlook 97 and select the *Tools* menu option followed by *Options* then deselect Excel from the *Journal* tab.

Application fails to run when workbook is first opened

Check that the *Auto_Open* name exists and refers to the **builder** macro.

Application runs more than once and ends with an error

Check all worksheets to ensure that the *Auto_Open* name does not have a reference to the worksheet. This can happen if *Sheet5* is copied, as an *Auto_Open* reference is copied with the new sheet name.

Report charts overlap or are too far apart

Check the following:

- That Excel is run in a maximized window.
- That the *Full screen* option in the *View* menu is not selected.
- That the screen resolution is set to 800 by 600.
- That the Formula Bar and Status Bar are displayed.
- That three rows of toolbars exist below the menu bar.
- That the Taskbar is at the bottom of the screen and is only as tall as the *Start* button.

"No data selected please check range of dates"

The *Report Date From* entry corresponds to a date for which performance collection data text files do not exist.

Application does not open the summary file correctly

Ensure the Windows Explorer settings are correct (described earlier under the heading *Windows Explorer Settings*).

Robert Russell (UK) © Xephon 1999

Bull has integrated WRQ's Reflection Signature Single Sign-On (SSO) software into its AccessMaster security management suite. AccessMaster's SSO lets users enter a password once and be authenticated to all authorized systems and applications.

Bull has also announced a new release of the SecurWare Internet security software suite. New features include encryption for building VPNs, data compression, and the NetWall V4 firewall. It runs on AIX and NT, with a Solaris version to follow soon. Out now, NetWall starts at US$1,850 for up 25 IP addresses, and SecurWare VPN starts at $5,000 per VPN device and $160 per user.

For further information contact:
Bull Information Systems, 2 Wall Street, Technology Park, Billerica, MA 01821, USA
Tel: +1 978 294 6000
Fax: +1 978 294 6440
Web: http://www.bull.co.uk

Bull Information Systems, Windsor House, 3-7 Albert Street, Slough SL1 2BH, UK
Tel: +44 1753 551554
Fax: +44 1753 705678

* * *

Lincoln Software has released a new version of its repository-based application development tool, Engineer for CICS, which now supports Java development. Users can now choose between HTML or Java front-ends, using CICS Gateway for Java or CICS Web Interface respectively. The product has facilities for application modelling and Java/HTML code generation. In addition to supporting CICS on AIX it supports development on System/390 mainframes and NT servers. It's out now, but no details on prices.

For further information contact:
Lincoln Software, 85 Liberty Ship Way, Suite 110B, Sausalito, CA 94965, USA
Tel: +1 415 339 1947
Fax: +1 415 339 1946

Lincoln Software Limited, Marlborough Court, Pickford Street, Macclesfield, Cheshire SK11 6JD, UK
Tel: +44 1625 616722
Fax: +44 1625 616780
Web: http://www.lincolnsoftware.com

* * *

IBM has announced that it's working with SCO and Sequent to develop a high-volume Unix for Intel's IA-64 chip. The new operating system is to be based on AIX and will also draw on technology from SCO's UnixWare and Sequent's PTX. In future the three companies are to de-emphasize their own brands of Unix in favour of the one unified brand. IBM also intends to transfer AIX technology to SCO's UnixWare to promote Unix in the IA-32 market.

For further information contact your local IBM representative.