40

AIX update

February 1999

In this issue

3 The syslog subsystem on AIX
8 Improving a DNS configuration
15 IPv6 – an overview
27 Understanding the sort command (2)
36 Process groups
38 Implementing a taskbar widget
50 Memory size display
51 Contributing to AIX Update
52 AIX news

AIX Update

Published by

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

North American office

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

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).

The syslog subsystem on AIX

INTRODUCTION

The AIX syslog subsystem is a client/server tool that makes it possible to handle all messages created by running processes in a homogeneous way.

Syslog comprises a server component, which actually manages incoming messages, and a library of functions that make it possible to send messages to the server from your own applications using the UDP protocol.

It's worth noting that most software available for AIX uses syslog to log its activity.

THE SERVER SIDE

The server component of the syslog subsystem is the **syslogd** executable file. This implements a daemon process that listens for incoming messages on a well-known UDP port. The configuration file for this daemon is */etc/syslog.conf*.

The commands to start and stop this subsystem are respectively:

```
startsrc -s syslogd [-a "[-d] [-s] [-f Confile] [-m MarkInt]"]
```

and:

```
stopsrc -s syslogd
```

Let's look at the two most important parameters to start the server:

[-s]	This optional parameter specifies that messages forwarded to other systems are in a 'shot' form.
[-f Confile]	This parameter is also optional, and specifies the configuration file to be used. The default value is */etc/syslog.conf*.

Other parameters are explained in the AIX documentation (the default values of these parameters are acceptable in normal systems).

You may also start and stop the syslog subsystem using either **smit** or **smitty**. If you want to know the subsystem's status, use the following shell command:

```
    lssrc -l -s syslogd
```

The syslog subsystem classifies all messages received into a number of categories, and the system administrator can handle each of these categories differently. Each category is identified by two values, known as '*facility*' and '*priority*'. The *facility* value identifies the category of the process that created the message, the priority value identifies the category of the message itself inside the specified facility. Here are the allowed values:

```
/*
** Facilities
*/
LOG_KERN          /* kernel messages */
LOG_USER          /* random user-level messages */
LOG_MAIL          /* mail system */
LOG_DAEMON        /* system daemons */
LOG_AUTH          /* security/authorization messages */
LOG_SYSLOG        /* messages generated internally by syslogd */
LOG_LPR           /* line printer subsystem */
LOG_NEWS          /* news subsystem */
LOG_UUCP          /* uucp subsystem */
LOG_CRON          /* clock daemon */
            /* other codes through 15 reserved for system use */
LOG_LOCAL0        /* reserved for local use */
LOG_LOCAL1        /* reserved for local use */
LOG_LOCAL2        /* reserved for local use */
LOG_LOCAL3        /* reserved for local use */
LOG_LOCAL4        /* reserved for local use */
LOG_LOCAL5        /* reserved for local use */
LOG_LOCAL6        /* reserved for local use */
LOG_LOCAL7        /* reserved for local use */

/*
**  Priorities
*/
LOG_EMERG         /* system is unusable */
LOG_ALERT         /* action must be taken immediately */
LOG_CRIT          /* critical conditions */
LOG_ERR           /* error conditions */
LOG_WARNING       /* warning conditions */
LOG_NOTICE        /* normal but signification condition */
LOG_INFO          /* informational */
LOG_DE    BUG     /* debug-level messages */
```

It's possible to customize the server by editing the configuration file */etc/syslog.conf* and other files specified with the **–f** option. Each line of this file comprises two parts: a *selector*, which determines the message class, and an *action*, which determines how you want the class to be managed. These two parts must be divided by one or more blanks or tabs. Here is the syntax of this configuration file taken from the on-line documentation:

```
Each line must consist of two parts:-

1) A selector to determine the message priorities to which the
   line applies
2) An action.

The two fields must be separated by one or more tabs or spaces.
Format:

<msg_src_list>    <destination>

where <msg_src_list> is a semicolon separated list of
<facility>.<priority>
where:

<facility> is:
   * - all (except mark)
   mark - time marks
   kern,user,mail,daemon, auth,... (see syslogd(AIX Commands
                                              Reference))

<priority> is one of (from high to low):
   emerg/panic,alert,crit,err(or),warn(ing),notice,info,debug
   (meaning all messages of this priority or higher)

<destination> is:
   /filename - log to this file
   username[,username2...] - write to user(s)
   @hostname - send to syslogd on this machine
   * - send to all logged in users.
```

It's important to note that you may redirect a message class to another system that's running the syslog subsystem. This may be useful if you want all messages to be collected on one machine.

When you've finished editing the configuration file for the server component, you have to create any files you plan to use that don't already exist and then refresh the server to apply any changes you have made. You can refresh the server by sending it a *HANGUP* signal using the **kill** command, or by stopping and restarting it.

If you want to communicate with the server from inside your own C/C++ software, you should examine the *syslog.h* file. This file contains declarations for all functions that allow your software to send messages to the syslog server process. The most important functions are: *openlog()*, *syslog()*, *closelog()*, and *setlogmask()*. Let's examine these functions in more detail.

- *openlog()*
  This function initializes communication between the user process and the server-side component of syslog. Note that you can omit it from your software. The syntax of this function is:

  ```
  void openlog(const char ID, int LogOption, int Facility)
  ```

  Where:

  – *ID* is a string that contains the header for generated messages

  – *LogOption* takes one of the following values:

  ```
  /*
   *  Option flags for openlog.
   *
   *    LOG_ODELAY no longer does anything; LOG_NDELAY is the
   *    inverse of what it used to be.
   */
  LOG_PID     /* log the pid with each message            */
  LOG_CONS    /* log on the console if errors in sending  */
  LOG_ODELAY  /* delay open until syslog() is called      */
  LOG_NDELAY  /* don't delay open                         */
  LOG_NOWAIT  /* if forking to log on console, don't wait() */
  ```

  – *Facility* is the 'facility' or process type. The value must be a member of the list of facilities presented earlier in this article.

- *syslog()*
  This function sends a message to the syslog's server-side component. Its syntax is:

  ```
  void syslog(int Priority, const char* Value, …)
  ```

  – *Priority* is the priority of the generated message. The value must be included in the list shown earlier in this article.

  - *Value* is a string that specifies the message format. Its syntax is similar to that used in the *printf()* function, the only difference being that you can use the identifier *%m* in the string to specify that the error message associated with *errno* is to be shown in the message.

- *closelog()*
  This function closes the communication channel to the server-side component of syslog. Its syntax is:

  ```
  void closelog()
  ```

- *setlogmask()*
  This function lets you set and modify the priority mask. By default, each message, along with its priority, is also a member of classes with a lower priority but the same facility.

  ```
  void setlogmask(int MaskPriority)
  ```

  - *MaskPriority* can take one of the following two values: *LOG_MASK(priority), LOG_UPTO(priority)*. The first value indicates that each message belongs only to its priority category. The second value, which is the default, indicates that each message belongs to its priority category and also to a category with lower priority values.

In a multithreaded environment, the following functions are available: *openlog_r()*, *syslog_r()*, *closelog_r()*, and *setlogmask_r()*.

Communication between a shell script and the server side component of syslog is possible using the **logger** command. The syntax of this is:

```
logger [-f File] [-i] [-p Priority] [-t Tag] [Message]
```

File is the name of a file containing a list of messages you want to be logged by syslog, *Tag* is a string used as the header of the message you are sending, *Message* is the message you want to be sent to the server. If you specify the **-i** option, the PID of the sending process is added to the message.

Marco Pirini
System Administrator (Italy)

Improving a DNS configuration

INTRODUCTION

Not that long ago, my company's TCP/IP network comprised only around forty clients and one server (which ran Oracle Database Server and Application Services). Since then, the network and network services have grown rapidly, and now comprise 700 local and remote clients and fifteen servers. This prompted us to switch from HOSTS file-oriented IP address resolution to DNS.

To this end, we set up one primary and one secondary DNS server, each located in its own physical network. Both DNS servers are multihomed (have more than one network interface), which allows clients to connect directly to the DNS server.

SAMPLE CONFIGURATION

The two subnets are 191.9.243.*xxx* and 191.9.244.*xxx*. We have primary DNS server '*Ramses'* and secondary DNS server '*Osiris'*. '*Anubis'* is an application server and '*papyrus'* is a network printer. *Ramses* is responsible for domain *Kairo*. Their IP addresses are:

```
     191.9.243.1  ramses
     191.9.243.60 papyrus
     191.9.243.2  anubis
     191.9.244.1  osiris
```

CONFIGURATION FILES ON PRIMARY DNS SERVER RAMSES

/ETC/NAMED.BOOT

```
directory       /etc
domain          kairo
primary         kairo                           named.data
primary         243.9.191.in-addr.arpa          named.191.9.243.rev
primary         244.9.191.in-addr.arpa          named.191.9.244.rev
primary         0.0.127.in-addr.arpa            named.local
```

/ETC/NAMED.DATA

```
@            9999999 IN   SOA     ramses.kairo. root.ramses.kairo. (
                                  1.207         ; Serial
                                  300           ; Refresh
                                  300           ; Retry
                                  3600000       ; Expire
                                  86400 )       ; Minimum
             9999999 IN   NS      ramses
loopback     9999999 IN   A       127.0.0.1     ; loopback (lo0)
localhost    9999999 IN   CNAME   loopback
anubis       9999999 IN   A       191.9.243.2
kerberos     9999999 IN   CNAME   anubis
papyrus      9999999 IN   A       191.9.243.60
osiris       9999999 IN   A       191.9.244.1
```

/ETC/NAMED.191.9.243.REV

```
@            9999999 IN   SOA     ramses.kairo. root.ramses.kairo. (
                                  1.207         ; Serial
                                  300           ; Refresh
                                  300           ; Retry
                                  3600000       ; Expire
                                  86400 )       ; Minimum
             9999999 IN   NS      ramses.kairo.
1       IN PTR ramses.kairo.
60      IN PTR papyrus.kairo.
2       IN PTR anubis.kairo.
```

/ETC/NAMED.191.9.244.REV

```
@            9999999 IN   SOA     ramses.kairo. root.ramses.kairo. (
                                  1.207         ; Serial
                                  300           ; Refresh
                                  300           ; Retry
                                  3600000       ; Expire
                                  86400 )       ; Minimum
             9999999 IN   NS      ramses.kairo.
1       IN PTR osiris.kairo.
```

/ETC/NAMED.LOCAL

```
@       IN      SOA     ramses.kairo. root.ramses.kairo.
        (
        1.1     ;serial
        3600    ;refresh
        600     ;retry
        4320000 ;expire after 50 days
        86400   ;minimum TTL of 1 day
        )
        IN      NS      ramses.kairo.
1       IN      PTR     localhost.
```

/ETC/RESOLV.CONF

```
domain kairo
nameserver 127.0.0.1
nameserver 191.9.243.1
```

CONFIGURATION FILES ON SECONDARY DNS SERVER OSIRIS

/ETC/NAMED.BOOT

```
directory   /etc
secondary   kairo                   191.9.243.1 named.data
secondary   243.9.191.in-addr.arpa  191.9.243.1 named.191.9.243.rev
secondary   244.9.191.in-addr.arpa  191.9.243.1 named.191.9.244.rev
primary     0.0.127.in-addr.arpa                named.local
```

/ETC/NAMED.191.9.243.REV

```
; zone '243.9.191.in-addr.arpa'   last serial 1000206
; from 191.9.243.1   at Fri Aug 10 16:14:25 1998

$ORIGIN 9.191.in-addr.arpa.
243     9999999 IN      SOA     ramses.kairo. root.ramses.kairo. (
                1000207 300 300 3600000 86400 )
        9999999 IN      NS      ramses.kairo.
$ORIGIN 243.9.9.191.in-addr.arpa.
1               IN      PTR     ramses.kairo.
2               IN      PTR     anubis.kairo.
60              IN      PTR     papyrus.kairo.
```

SAMPLE CLIENT CONFIGURATION FROM ANUBIS

/ETC/RESOLV.CONF

```
domain kairo
nameserver 191.9.243.1
nameserver 191.9.244.1
```

CONFIGURING THE RESOLVER

/etc/netsvc versus NSORDER

AIX uses several services for resolving host names. A default setting is used to determine the order in which these services are tried for resolving host names and IP addresses. The default order can be overwritten using the configuration file */etc/netsvc.conf* (if the file doesn't already exist, create it). Specify the desired order using the entry:

```
     hosts = value [, value]
```

where *value* can be {*bind*|*local*}.

- *bind* uses BIND/DNS services for resolving names.
- *local* searches the local */etc/hosts* file for resolving names.

Notice that the *NSORDER* environment variable overrides the settings in the */etc/netsvc.conf* file.

SAMPLE /ETC/NETSVC.CONF

```
hosts = bind
```

NSLOOKUP TEST ON ANUBIS

Below are the contents of *nslookup* on host Papyrus.

NSLOOKUP

```
Server:  ramses.kairo
Address:  191.9.244.1

Name:    papyrus.kairo
Address:  191.9.243.60
```

After our company established a connection to the Internet (protected through a firewall system), we adjusted clients' *resolv.conf* files to include the IP address of our ISP's DNS server.

/ETC/RESOLV.CONF

```
domain kairo
nameserver 191.9.243.1
nameserver 191.9.244.1
nameserver 194.196.47.2
```

At that time we did not wish to modify our DNS server set-up, as this was running without any problems. However, the set-up has the following drawbacks:

- If the primary DNS server is down for maintenance, clients that access the server by means of their */etc/resolv.conf* files experience poor **login**, **telnet**, and **ftp** performance as a result of the long timeout that occurs before the resolver contacts the secondary DNS server, which then takes over name resolution.
- If our ISP's DNS server is not available, none of our clients can access the Internet as none of the other DNS servers can resolve Internet names.

The solution for improving performance and availability is:

- Extend the DNS file *named.boot* so that the DNS server asks for Internet address resolution from the ISP's DNS server. The internal DNS server is then set to cache this information.

  This requires the following additional lines in the file */etc/named.boot* on *Ramses*:

  ```
  cache          .     named.ca
  forwarders     194.196.47.2  194.196.47.4

  Contents of the /etc/named.ca file on RAMSES

  .        3600000 IN NS ns1.ibm.co.at.
  ns1.ibm.co.at.  3600000   A  194.196.47.2
  .        3600000 IN NS ns2.ibm.co.at.
  ns2.ibm.co.at.  3600000   A  194.196.47.4
  ```

  This adjustment to our internal DNS server means that addresses that have already been resolved are not sent for resolution to the ISP's DNS server until their Time To Live (TTL) is exceeded. This reduces traffic on the leased line to the ISP and makes us less dependent on the ISP's DNS server.

- Install a caching only DNS server on the heavily used application server *Anubis*. This requires changes to the following files:

/ETC/RESOLV.CONF

```
domain kairo
nameserver 127.0.0.1
nameserver 191.9.243.1
nameserver 191.9.244.1
```

/ETC/NAMED.BOOT

```
directory       /etc
primary         0.0.127.in-addr.arpa          named.local
cache           .                             named.ca
forwarders      191.9.243.1 191.9.244.1 194.196.47.2
```

/ETC/NAMED.CA

```
.        3600000 IN NS  ramses.kairo.
ramses.kairo.   3600000   A  191.9.243.1
.        3600000 IN NS  osiris.kairo.
osiris.kairo.   3600000   A  191.9.244.1
.        3600000 IN NS  ns1.ibm.co.at.
ns1.ibm.co.at.  3600000   A  194.196.47.2
```

/ETC/NAMED.LOCAL

```
@       IN      SOA     anubis.kairo. root.anubis.kairo.
        (
        1.1     ;serial
        3600    ;refresh
        600     ;retry
        4320000 ;expire after 50 days
        86400   ;minimum TTL of 1 day
        )
        IN      NS      anubis.kairo.
1       IN      PTR     localhost.
```

Modify the */etc/rc.tcpip* run command script to start the **named** server process automatically after a reboot. Uncomment the line *#start /usr/sbin/named "$src_running"*, or use the **smit** interface to perform this task.

With just this improvement, the application server loses its dependence on the availability of the ISP's DNS server as, after just three hours of execution in a production environment, all required TCP/IP aliases are already resolved and available locally at the caching DNS server.

You can check the DNS caching-only server by sending an ‘INT’ signal to its **named** process. This produces a dump of the current address table to */var/tmp/named_dump.db*.

While these modifications solved our previous problems, they also introduced some new ones.

1 Networks already defined in the **named.data** file (which resolves TCP/IP aliases to IP addresses) suffer *very* poor login performance when they start **telnet** or **ftp**.

2 If the Internet connection fails, the problem increases dramatically.

WHAT’S HAPPENING?

After a lot of debugging and investigation we found the answer to this problem. We failed to adjust the name server’s address-to-name mappings files and *named.boot* file.

Every request from a client on a network that is not mentioned in the *named.boot* file to an application server configured using DNS results in a request to the DNS server to find the client’s identity. The DNS server is unable to service this request, as it is not configured correctly. Instead it just forwards the request to DNS servers on the Internet – when the Internet is available – and eventually the internal DNS server gets a reply to say that there is nobody available to resolve the address to a name. The internal DNS server then forwards this information to the application server, which then completes the **telnet** or **ftp** login process.

When either the Internet or just the Internet DNS server is not available, the reply to the reverse name resolution query takes even longer but is essentially the same – no server is available to resolve the query.

THE SOLUTION

For each network in the *named.data* file, you have to add a line for the reverse address resolution to the *named.boot* file. Thus, for instance, network *191.9.242.xxx* requires a file with reverse address resolution information such as *named.191.9.242.rev*.

```
    primary   242.9.191.in-addr.arpa   named.191.9.242.rev
```

CONCLUSION

DNS administration is not as simple as it seems. If you can **nslookup** an alias successfully, it doesn't mean that the address can be resolved to the name also. For a detailed discussion of DNS and BIND, consult *Help for Unix System Administrators* by Paul Albitz and Cricket Liu.

Michael Imhotep (Australia)

IPv6 – an overview

INTRODUCTION

The recent and rapid growth in the number of devices connected to the Internet has highlighted a significant problem with the current Internet Protocol, IPv4. IPv4 uses 32-bit addresses, providing a little over four billion discrete addresses. At first sight this would seem sufficient for current demand, as it is several orders of magnitude greater than the current number of systems connected to the Internet. The problem is related to the manner in which IPv4 addresses are structured.

An IPv4 address comprises two parts: a network identifier and an identifier of the host system on this network. The effect of this structure is that, once a network identifier has been allocated, all the host addresses in that network are taken from the IPv4 address space, even if there are in fact only one or two nodes connected to the network. The result is a large number of 'lost' addresses. The problem is compounded by the requirement that IPv4 addresses must be globally unique, even if the network to which the nodes are attached is not connected to the outside world, the Internet. Further, the generalization of the Internet into domestic and large-scale, multi-client applications is generating new applications and devices that will use the Internet as their means of communication. These include television set-top boxes (for cable or satellite TV), automatic teller machines (ATMs), cash registers, and other forms of 'electronic point of sale' (EPOS) device. These new devices and applications will only accelerate the demand for new IP addresses.

The end result is that the current IPv4 address space will run out some time between the years 2000 and 2010, depending on whose projection you choose to use. It is therefore essential that a new addressing scheme be decided upon and implemented in the very near future. IPv6, previously known as 'IPng' (*IP next generation*), provides this addressing scheme. It is the outcome of numerous meetings and experiments used to agree upon the optimum IP address structure for the future.

While the address space exhaustion problem was the driving force behind the development of IPv6, a number of other shortcomings of the IPv4 specification were identified and addressed in the new standard. These include:

- *Improved efficiency*. A number of performance optimizations and packet handling improvements have been made to reduce the number of calculations as IP packets make their way through the various networks between the sender and receiver.

- *Service classes*. These are used to identify the transmission requirements for a data exchange. These can be compared to the Quality of Service (QoS) parameter of link-layer protocols, such as ATM.

- *Multicasting*. Multicasting involves simultaneously sending a message to a number of different nodes. The IPv4 implementation, known as 'broadcast', has been the source of a number of administration and performance difficulties. IPv6 provides a multicast mechanism which is fully integrated with the other addressing schemes, and which should offer a much more flexible and easier to use solution.

- *Security*. The adoption of the Internet by commercial, financial, and medical institutions has highlighted the need for a means of ensuring the confidentiality of the data that is sent across public networks. IPv6 introduces two new security services: authentication and encryption. Authentication is used to ensure that the person with whom you are communicating is who they claim to be, and is also used to prove that you are who you say you are. Encryption 'scrambles' the transmitted data so that, if a third party were to intercept the message, they would not be able to read it. Only those parties that hold the 'keys' necessary to decrypt the message are able to read it.

In addition to the above improvements, the Internet Engineering Task Force (IETF), which controls the Internet specifications and standards, placed a number of other requirements on the new IP standard. Perhaps the most important of these is the ability to coexist and interoperate with IPv4 nodes and networks, and to offer a transition path from the old to the new protocol suite.

The implementation of IPv6 for AIX 4.3 was performed by Groupe Bull in conjunction with the INRIA.

BACKGROUND

This section introduces some of the networking ideas necessary to understand the following discussions.

What is the Internet Protocol (IP)?

IP, the Internet Protocol, as its name suggests, is a communications protocol suitable for both connecting nodes on a network and providing a link between different networks. In communications jargon, it is described as being an 'unreliable', 'connectionless' delivery system. It is unreliable in that the protocol is allowed to lose or 'drop' packets of data (datagrams) in the event of congestion or other problems. The fact that it is unreliable means that the responsibility for ensuring reliable data transfer is placed on the transport protocol or other higher-level protocols, such as TCP (Transmission Control Protocol), which runs on top of IP.

Although IP is classed as an unreliable delivery mechanism, within the confines of (say) an Ethernet-based Local Area Network (LAN) very few data packets are lost. It is only when the communicating systems are separated by large distances and/or very busy networks that data loss becomes a significant problem.

IP is connectionless in that each packet of data sent on the network contains the address of the destination system, in the same way that letters in the postal system are all contained in addressed envelopes. A connection-oriented system, in contrast, is like the telephone system where the caller addresses the receiver only once, at the outset of the communication, and a communication channel is opened – the telephone rings, and the receiver picks up the handset. This communication channel remains open for the duration of the conversation, and is closed when communication terminates.

Terminology

We have already seen a number of networking terms, such as node and datagram, in the preceding text without them being precisely defined. In the rest of this article I shall take the terminology from the IPv6 specifications. These are:

- *Node*. A device on a network. In the IPv6 specification this is a device that implements IPv6, but I shall use it for both IPv4 and IPv6 devices.
- *Router*. A router is a node that forwards packets not addressed to itself. In IPv4, this device was referred to as a ‘gateway’.
- *Host*. A node that is not a router, that is, a node that does not forward packets not addressed to itself. This is usually an end-user system or application server.
- *Link*. The medium over which nodes communicate. This includes the physical medium (cable, radio, infra red, etc) and the base or link-level protocol, such as Ethernet or ATM.
- *Interface*. The connection between a node and a link. A node may have more than one interface – a router, for example, must have at least two.

- *Neighbour*. Nodes connected to the same link are said to be neighbours.
- *Packet*. The elementary unit of data sent between two communicating nodes. This is known more generally as the Protocol Data Unit (PDU). An IPv6 PDU comprises a header and a payload. This is the same as an IPv4 datagram.

The IP packet

As described in the above definitions, the IP packet, or PDU, comprises two parts: the header and the payload (the top part of Figure 1).

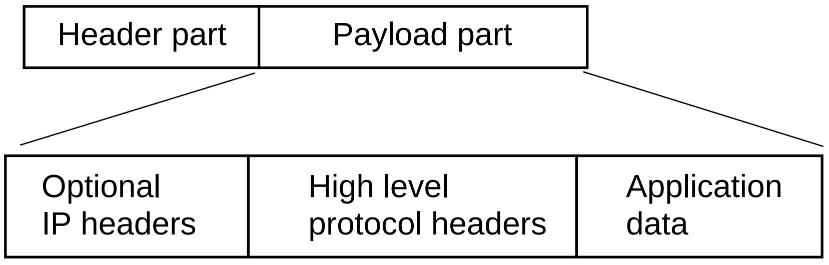

Figure 1: An IPv6 datagram (top) and payload (bottom)

The header contains all the information required by the protocol to deliver the payload to the destination. Continuing the postal-service analogy, the envelope, with the address and stamp, are the header and the letter contained in the envelope is the payload. In general, it is not necessary (indeed it's often illegal) to examine the contents of the envelope/payload to determine where the letter or packet should be delivered.

All IP packet headers start with a four-bit version number (not surprisingly, '4' for IPv4 and '6' for IPv6). This version number also defines the structure and content in the remainder of the header. The other header information includes things such as source and destination addresses and packet size.

The content of the payload is not defined by IP, except that it may contain optional header information. In general, higher-level protocol headers follow any optional IP headers, along with the application data. This is shown in the bottom part of Figure 1.

IPV6 COMPARED WITH IPV4

The two most striking differences between IPv4 and IPv6 are the extended 128-bit address fields and the greatly simplified packet header format. IPv4's variable-length header format, which is responsible for a number of performance problems in protocol handling, has been replaced by a fixed size header in IPv6. The IPv6 header is shown in Figure 2 below. As can be seen, it is made up of ten four-byte words, giving a total header length of 40 bytes.

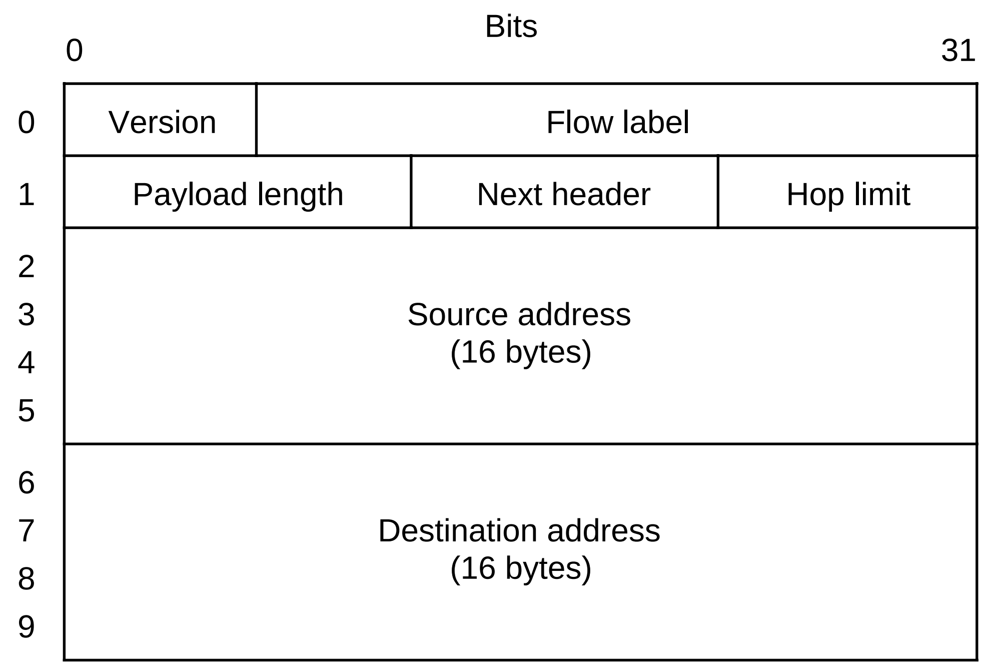

Figure 2: IPv6 packet structure

The meaning of the different fields is given below:

- *Version*. The IP version number – '4' for IPv4 and '6' for IPv6.
- *Prio*. Packet delivery priority.
- *Flow label*. Used for Quality of Service and caching. Routers may maintain a context associated with a *Flow Label* in order to optimize performance.
- *Payload length*. Length of the payload section. This is a 16-bit field. If the payload is longer than 64 KB (2^{16}), then this field

contains zero and the payload length is given in an extension header (see below).

- *Next header*. Identifies the type of the optional extension header (see below) that follows this fixed header. A value of '59' indicates that there is no extension header.

- *Hop limit*. A hop is the transit of a packet between two routers (which have interfaces on the same link). The hop-limit stipulates the maximum number of hops that a packet may take before being discarded. It is a mechanism used to prevent the networks becoming clogged by packets in infinite routing loops. The hop limit is set by the sending node and decremented at each router. If the value reaches zero, the packet is discarded. IPv4 headers contain a 'Time-To-Live' (TTL) field that is essentially the same, as the value in this field is also decremented at each hop, and not at each second as the name suggests.

- *Source address*. 128-bit address of the node (more precisely, the 128-bit address of the interface, as the node may be connected to more than one network, for instance, if it acts as a router) from which the packet originated.

- *Destination address*. The address of the interface or interfaces to which the packet is destined.

It's worth noting that checksums have been completely eliminated from IPv6. It's now the responsibility of higher-level protocols (principally TCP and UDP) to ensure data integrity.

Packet sizes

As a message transits various networks between the source and destination nodes, the different underlying network technologies may impose different maximum packet lengths, or maximum transmission units (MTUs). In IPv4, each time a packet reaches a network that has a smaller MTU size than the current packet size, the packet is split, or fragmented, into two or more smaller packets that fit inside the MTU of the next network. IPv6 handles the problem differently by determining the smallest maximum transmission unit size of all the networks along the route between the source and destination nodes. In this way fragmentation is the sender's responsibility. This eliminates the considerable overhead of fragmentation handling in intermediate routers. In IPv6 fragmentation information is carried in an extension header, and this information is used only at the destination host and not by intermediary nodes.

Extension headers

When the standard header is too small to accommodate all the necessary header information associated with the packet, IPv6 uses additional extension headers that make up the first part of the payload. We have already seen a couple of instances where extension headers are necessary – to handle payloads greater than 64 KB and to carry fragmentation information. Other examples are for IP control data, routing and security information, and data pertaining to a higher-level protocol, such as TCP, UDP, or FTP.

It can be seen from Figure 3 that the type of the extension header is not given in the header itself, but rather in the *Next Header* field of the preceding header. In this way, extension headers can be chained one after the other. If more than one extension header is used, then the extension headers must be sent in a pre-defined order.

IPV6 ADDRESSING

IPv6 addressing uses 128-bit (16 byte) addresses. As discussed in an earlier *AIX Update* article on 64-bit computing, 2^{64} is a huge number, an order of magnitude greater than the number of seconds since the beginning of the universe. 2^{128} is the square of 2^{64} and it is difficult to find a number in the physical world that comes close to this figure. The number of individual molecules in the solar system is the sort of number that we're talking about. With such a large number of possible addresses, it's difficult to imagine the day when we will run out again, but I guess that's what the original designers thought when they chose 32-bit addresses.

The lifetime of IPv6 addresses

IPv6 addresses are allocated for a fixed period of time. The default value is 60 hours, but this may be extended to 'forever'. When specifying the lifetime of an address two values are given:

1 The duration for which the address is ‘preferred’

2 The duration for which the address is ‘valid’.

When an address is in the preferred state, applications may use it to establish communication with other nodes. When the preferred state time has elapsed, the address becomes ‘depreciated’. In this state it is not possible to establish new communications using this address, though applications already using it can continue to do so. After the ‘valid’ time has elapsed, the address becomes ‘invalid’ and is unusable by all applications.

This mechanism provides a technique for gracefully renumbering interfaces and shutting down services.

IPv6 address notation

While IPv4 addresses are written as a series of four decimal numbers separated by a dot (‘.’), for example ‘192.90.72.2’, IPv6 uses eight groups of four hexadecimal characters separated by a colon (‘:’), for example:

```
5EA1:2AB0:F397:11A6:9876:5432:1FED:CBA9
```

Leading zeros may be omitted:

```
5EA1:2AB:39:6:0:0:0:CBA9
```

A double colon (‘::’) may be used to represent a consecutive sequence of zeros – many IPv6 addresses will contain long strings of zeros as a result of the way in which addresses are allocated. For example, the above address may be written as:

```
5EA1:2AB:39:6::CBA9
```

To avoid ambiguity, the double colon may appear only once in an address.

While it may be useful to be able to recognize and understand IPv6 addresses, it’s unlikely that you’ll have to work with them directly unless you are a system or network administrator – you’ll continue to use the symbolic names, such as *www.frec.bull.com,* as you do today.

As an aside, the use of the colon as an address separator in IPv6 conflicts with its use in separating the IP port number from the address in a World Wide Web (WWW) Universal Resource Locator (URL). Overloading this character results in ambiguity. The proposed solution is that numeric IPv6 addresses will not be allowed in URLs.

IPv6 address space partitioning

With such a large number of available addresses (2^{128} is approximately 10^{39}), it is possible to subdivide the IPv6 address space in a way that is more flexible and efficient than is possible with IPv4. Some of the more important address types are discussed in the following sections.

An important characteristic of IPv6 addresses is that the 32-bit IPv4 addresses are a subset of this space. This means that it is not necessary for IPv4 nodes and networks to be assigned new addresses to work with IPv6. This ensures that the transition to IPv6 produces as little disruption as possible to existing networks.

Unicast or Point-to-Point addresses

This is the most common of all addressing modes, in which an application on one host communicates with an application or user on another host, in a direct one-to-one relationship.

There are a number of special unicast addresses, including:

- *Unspecified address*. This address is used by a node during its initialization phase, prior to obtaining its proper address. Its value is ‘0:0:0:0:0:0:0:0’, which can be abbreviated to ‘::’.
- *Loopback address*. This address is equivalent to IPv4’s ‘127.0.0.1’ loopback address. It is used by a node to address a packet to itself. As it is a loopback address, no packet transiting a network will ever have this address. Its value is ‘::1’.
- *Link local address*. A link local address limits the scope of the address to the link to which it is attached. This limits communication to the node’s neighbours (see definitions above).
- *Site local address*. A site local address limits the validity of an

address to the group of networks contained in a single geographical site. Although this is an interesting idea, it presents a number of technical difficulties, and as such it is unlikely to be used or implemented at many sites.

- *IPv4 mapped addresses*. These are IPv4 addresses mapped to the IPv6 address space. This allows IPv4 applications to communicate using IPv6 addresses. The mapping is performed by the sending and receiving hosts, and as such these addresses never transit on the network. These addresses have the form '::FFFF:a.b.c.d', where 'a.b.c.d' is an IPv4 address.

- *IPv4 compatible addresses*. These addresses are used to send IPv6 packets over an IPv4 network. This technique is known as 'tunnelling', and will be described later. These addresses have the form '::a.b.c.d'.

Multicast addresses

Multicast addresses replace the IPv4 'broadcast'. They allow a host to send a packet to several other hosts simultaneously in an efficient manner. All the nodes that are involved in a multicast operation join a 'multicast group'. In doing so, they are assigned a multicast address that contains a 'multicast group identifier' – different groups get different identifiers. When a node sends a message (packet) to this address it is received by all nodes in the group. Some group identifiers are predefined in the IPv6 standard, for example for Network Time Protocol (NTP) and Dynamic Host Configuration Protocol (DHCP) servers.

Anycast addresses

Anycast addresses introduce a new concept to IP: they are a mixture of unicast and multicast. The idea is that a node can send a packet to a group of other nodes but only one of them will actually receive it. The frequently cited example for this type of address is that of a name server, where a node can send a request for name resolution to a generic group of name servers, but only the nearest (in a network sense), will receive, treat and reply to the request.

SECURITY

As discussed in the introduction, IPv6 introduces two new security services: authentication and data confidentiality. Authentication enables nodes to prove that they are who they say they are. This issue is one of the security problems that have not been resolved in IPv4.

Data confidentiality involves encrypting the contents of the payload such that it is indecipherable by any third party that might intercept the packet.

Since authentication does not involve issues of confidentiality, it should not be difficult to obtain general acceptance. However, data confidentiality and encryption are words that induce nervous twitches in some parts of the world, where governments want to be able to read the e-mail of anyone thought to be involved in subversive activities. Because of this, a universal and interoperable mechanism for sending confidential data may still be some time away.

Today's security systems operate using 'keys' that are used for both authentication and encryption. It is not sufficient just to send encrypted data across the network to another person – if the recipient is to be able to read the message, they also need the right key to decipher it. The distribution of keys among communicating parties is outside the scope of IPv6, but the specification is based on accepted practices and new research. Security information, such as the encryption algorithms used, can be carried by the security related extension headers.

CONCLUSION

In a future article I will describe some practical issues associated with IPv6, such as interoperability with and migration from IPv4. I will also discuss the network applications and configuration tools for IPv6 in AIX 4.3.

Jez Wain
System Architect
Groupe Bull SA (France)

Understanding the sort command (2)

This month's instalment concludes this article on the ***sort*** *command.*

SORT KEYS

Databases with numerous rows and columns are the regular stuff of sort operations, so there's a need to be able to specify the range of columns to use in sorting. A sort key is used to specify the fields and (optionally) the column numbers the **sort** command uses when making comparisons.

The default field separator for ASCII sorts is the 'blank character', which can be a series of spaces and tabs. One of the benefits of being able to sort on fields rather than character by character across the line is that columns don't have to 'line up'.

Multiple sort keys can be specified in a single sort operation, and they are processed serially. In other words, a sort is performed using the first key, then results with equal values using the first key are passed through another sort operation using the second key, etc.

The two methods of specifying sort keys are the 'skip' method and the **-k** flag method.

The 'skip' method

The list below summarizes the syntax of the skip method.

```
+FSkip.CSkip[Modifier]  -FSkip.CSkip[Modifier]
```

+FSkip	Fields that are skipped to reach the first field of the sort key (the default is the beginning of the line).
.CSkip	Characters within *+FSkip* skipped to reach the sort key (the default is none).
Modifier	An optional modifier for *+FSkip.CSkip*.
-FSkip	Fields skipped to reach the first field *after* the sort key (the default is the end of the line).

.CSkip	Characters within *-FSkip* that are skipped to reach the first character *after* the sort key (the default is none).
Modifier	An optional modifier for *-FSkip.CSkip*.

The skip method specifies the fields and columns to skip to reach the beginning and (optionally) end of the sort key. First, let’s examine skipping fields.

SKIPPING FIELDS

Suppose the first three entries in a database called *names.database* are as follows:

```
Jackson, Fredrick   Age=30   Department=B90   Occupation=Developer
Johnson, Albert   Age=30   Department=B10   Occupation=Manager
Abramson, Joseph   Age=29   Department=J90   Occupation=Designer
```

Using the sort key **+1** tells **sort** that the first field (‘Jackson’, in the first record) is to be skipped. The remainder of the line (‘Fredrick’ onwards) is the text that **sort** is to use for comparisons.

Entering **sort +1 names.database** results in the following:

```
Johnson, Albert   Age=30   Department=B10   Occupation=Manager
Jackson, Fredrick   Age=30   Department=B90   Occupation=Developer
Abramson, Joseph   Age=29   Department=J90   Occupation=Designer
```

This output represents text sorted alphabetically beginning with the data following the first field on each line.

If **+2** were specified as the sort key, everything from ‘Age’ on would be considered, as the first two fields are skipped.

Entering **sort +2 names.database** would result in the following output:

```
Abramson, Joseph   Age=29   Department=J90   Occupation=Designer
Johnson, Albert   Age=30   Department=B10   Occupation=Manager
Jackson, Fredrick   Age=30   Department=B90   Occupation=Developer
```

The output is the text sorted alphabetically starting with the third field on each line (the *Age* field). Note that the entries for ‘Johnson’ and ‘Jackson’ have the same data for *Age* (‘30’). In this case, **sort** looks at the data in the next field (*Department*) to determine the order. As

'B10' comes before 'B90' alphabetically, 'Johnson' precedes 'Jackson'.

You can specify the end field for the sort key if you don't want the rest of the line to be used for sorting. Suppose you want to sort using only *Age* and *Department* in the above example. The sort key for this would be **+2 -4**. The **+2** tells **sort** to skip the first two fields and start sorting on the third, and **-4** tells it to stop sorting after the fourth field.

SKIPPING COLUMNS WITHIN FIELDS

The sort key can be further qualified. Suppose database *budget.database* contains the following text:

```
tax file = data.94tax
tax file = data.84tax
tax file = data.93tax
insurance file = data.94insur
insurance file = data.84insur
insurance file = data.93insur
receipt file = data.94rcpt
receipt file = data.84rcpt
receipt file = data.93rcpt
```

If you want to sort starting with the first two digits after the period ('.') in the file name, use the sort key **'+3.6'**, as in:

```
sort +3.6 budget.database
```

This returns the following:

```
insurance file = data.84insur
receipt file = data.84rcpt
tax file = data.84tax
insurance file = data.93insur
receipt file = data.93rcpt
tax file = data.93tax
insurance file = data.94insur
receipt file = data.94rcpt
tax file = data.94tax
```

The **+3.6** tells **sort** to skip three fields, and then six columns. For example, in the line:

```
tax file = data.94tax
```

The first field is 'tax', the second is 'file', and the third is the equals sign ('='). 'data.94tax' is the first field to sort on.

Column numbering counts leading spaces as part of the field for all but the first field in a string. This can easily cause confusion – although **sort** considers leading blanks in the first field as ASCII characters for comparisons, you do not number the leading blanks if you specify the first field as part of the sort key. Therefore, even though the period is the fifth character in the field **data.94tax**, the **.6** tells **sort** to skip the leading blank, the four characters 'data', and the period to get to the first character of the sort key, the '9'.

You can specify an ending column for a sort key as well. This is the *.CSkip* postfix to the *-FSkip* parameter.

ADDING MODIFIER INFORMATION TO THE SKIP METHOD

The skip method can be enhanced even further by supplying a modifier to the skip parameters. The modifiers are letters that mimic the behaviour of their flag counterparts. One or more of the following letters can be used as modifiers: **b**, **d**, **f**, **i**, **n**, and **r**. The list below summarizes the letters' effect as modifiers.

b	Disregard leading blanks
d	Sort in dictionary order
f	'Fold' letters to upper case (ignore case)
i	Ignores non-printing characters
n	Perform a numeric sort
r	Reverse the order of the sort results.

For example, entering **sort +3.6r budget.database** results in the following:

```
tax file = data.94tax
receipt file = data.94rcpt
insurance file = data.94insur
tax file = data.93tax
receipt file = data.93rcpt
insurance file = data.93insur
tax file = data.84tax
receipt file = data.84rcpt
insurance file = data.84insur
```

The **r** in this example indicates that the data in the sort key is to be displayed in reverse order. AIX's **sort** command is very powerful, and, consequently, sort keys can be quite complex.

THE -K FLAG METHOD

The **-k** flag method is more intuitive and, as a result, more commonly used than the skip method. Where the skip method tells the **sort** command how many fields and columns to skip, the **-k** flag method indicates the sort keys explicitly.

The list below summarizes the syntax of the **-k** flag method of defining a sort key. Note that the same modifiers apply to the **-k** flag method as to the skip method.

```
     -k FStart.CStart[Modifier],FEnd.CEnd[Modifier]
```

-k	The sort key flag.
FStart	The first field of the sort key (the default is the beginning of the line).
.CStart	The column within *FStart* for the beginning of the sort key (the default is the first column in the field).
Modifier	An optional modifier for the starting field.
FEnd	End field of the sort key (the default is the end of the line).
.CEnd	The column within *FEnd* for the ending of the sort key (the default is the last column of the field).
Modifier	An optional modifier for the end field.

SPECIFYING FIELDS

You can use the **-k** flag to indicate specific fields to use as a sort key. Suppose a database called *dates.database* contains the following:

```
30 January 1929 data
26 February 1938 data
22 March 1947 data
18 April 1956 data
14 May 1965 data
10 June 1974 data
```

```
06 July 1983 data
02 August 1992 data
```

The database is currently arranged in calendar order by month without regard to day or year. If you were to enter:

```
sort -k 2 dates.database
```

the results would be as follows:

```
18 April 1956 data
02 August 1992 data
26 February 1938 data
30 January 1929 data
06 July 1983 data
10 June 1974 data
22 March 1947 data
14 May 1965 data
```

SPECIFYING COLUMNS WITHIN FIELDS

The way to specify a column for a sort key is similar to that of the skip method in that the column number follows a period after the field number. Remember that leading blanks in all but the first field are considered part of the field, unless you use the **b** modifier in the sort key. For example, if you enter:

```
sort -k 3.4b dates.database
```

the results are:

```
02 August 1992 data
06 July 1983 data
10 June 1974 data
14 May 1965 data
18 April 1956 data
22 March 1947 data
26 February 1938 data
30 January 1929 data
```

The sort key **3.4b** tells the **sort** command to consider everything from the fourth non-blank character of the third field to the end of the line as the sorting criterion.

SPECIFYING START AND END COLUMNS

As with the skip method, you can specify start and end columns with the **-k** flag. For example, suppose *joblog.database* contains the following:

```
010798-SMI-0001  Roger Smith, general inspection.
020698-JON-0002  Jones.  Repaired valve, inspect.
030898-WIL-0003  Paul Williams, G.I.
040598-MAR-0004  Bill Martin inspected drive train, adj.
050798-JON-0005  Jones.  General Inspection.
060698-WIL-0006  Paul Williams, drive train adjust.
070398-SMI-0007  Roger Smith, replace master gear.
```

Each entry starts with a fifteen character string containing a date, the first three characters of the name, and a sequence number. The remaining information is a log of activity by service personnel. The list is sorted by sequence number and, coincidentally, by date. Suppose you want the list sorted alphabetically by the three characters between the date and the sequence number in the first field of each line. To do this, enter:

```
sort -k 1.8,1.10 joblog.database
```

which results in the following:

```
020698-JON-0002  Jones.  Repaired valve, inspect.
050798-JON-0005  Jones.  General Inspection.
040598-MAR-0004  Bill Martin inspected drive train, adj.
010798-SMI-0001  Roger Smith, general inspection.
070398-SMI-0007  Roger Smith, replace master gear.
030898-WIL-0003  Paul Williams, G.I.
060698-WIL-0006  Paul Williams, drive train adjust.
```

In this case, **1.8,1.10** represents a sort key from the eighth to the tenth character of the first field.

SOME EXERCISES

Here are three examples of the sort command in practice.

EXAMPLE 1

A database called *access.log* contains names of department staff. The log is written by an automated program that tracks access into a secure environment based on the user’s account name. The database contains mixed case entries, some that have titles in front of the names and some entries that have a last name followed by a first name. Some entries contain preceding spaces before the data. The raw data, if displayed, would appear as follows:

```
administrator Martin
coordinator Williams
  Smith, Edwin
  Smith, Edwin
  Baker, Mark
administrator Jones
coordinator Williams
  Baker, Mark
```

If you were to simply enter **sort access.log**, the result would be:

```
  Baker, Mark
  Baker, Mark
  Smith, Edwin
  Smith, Edwin
administrator Jones
administrator Martin
coordinator Williams
coordinator Williams
```

Note that the command sorts each line alphabetically on the first word, considering blank spaces as alphabetic characters. Suppose you want to sort the database alphabetically on the first word, ignoring leading blanks and the case of the entries, and that you also want to ignore duplicate entries. How could you accomplish this?

The way to do this is to use the **-f** flag to 'fold' all entries to uppercase prior to the sort operation, the **-b** flag to ignore leading blanks, and the **-u** flag to disregard all but one of each duplicate entry. The command:

```
sort -f -b -u access.log
```

yields the following results:

```
administrator Jones
administrator Martin
  Baker, Mark
coordinator Williams
  Smith, Edwin
```

EXAMPLE 2

A database called *account.log* contains entries with fields separated by colons (:). Each entry has the following format:

```
inv:name:date:cust
```

inv is the invoice number, *name* is the employee name, *date* is the service date, and *cust* is the name of the customer. The database contains the following records, which are sorted alphabetically by employee name:

```
0055A:adams:052298:ABC Company
0030B:baumont:032998:XYZ Corp.
0095D:daniels:100597:XYZ Corp.
0050J:jones:051798:DATACON
0090M:martin:102597:ABC Company
0033R:robertson:030398:MATTERLY Co.
0075S:smith:081698:XYZ Corp.
```

If the database needs to be sorted on the date field, what **sort** command would accomplish this? The answer is first to use the **-t** flag to specify the colon as the field separator. Then use sort keys to indicate the field on which to sort. In this case, the database should be sorted on the third colon-delimited field (*date*).

One might assume that it's sufficient to specify the third field as the sort key. However, a default sort of the date fields above would put some dates in 1998 before others in 1997. Therefore, multiple sort keys need to be used. To sort using a six-digit date field, start with the last two digits (the year), then the first two (the month), and finally the middle two (the day).

The following **sort** command:

```
sort -t: -k 3.5,3.6 -k 3.1,3.2 -k 3.3,3.4 account.log
```

result in the following:

```
0095D:daniels:100597:XYZ Corp.
0090M:martin:102597:ABC Company
0033R:robertson:030398:MATTERLY Co.
0030B:baumont:032998:XYZ Corp.
0050J:jones:051798:DATACON
0055A:adams:052298:ABC Company
0075S:smith:081698:XYZ Corp.
```

Note that all entries are now sorted in order by date. It's important to note that, unlike the default blank separator, a field separator specified using the **-t** flag is *not* counted as part of the field.

David Chakmakian (USA) © Xephon 1999

Process groups

If you have an application that uses SNA to communicate to a remote host, then the following article may prove invaluable to you. In my environment, we use IBM's DirectTalk/6000 for customer queries via a telephony interface. Within the DirectTalk/6000 application, it is possible to write custom applications that can communicate with remote nodes. I wrote one such application that communicates, via an SNA connection, to a remote OS/390 host. Each weekend, the OS/390 host is taken through an IPL. It was during this IPL on the OS/390 host that some problems arose. My application was written to continue to poll the SNA connection until connectivity is re-established, so I thought I had addressed the issue of the SNA outage during the IPL.

However, shortly after the loss of SNA connectivity, my applications would terminate. Not only did they terminate, but various other applications under the DirectTalk/6000 umbrella failed as well. A little diagnosis uncovered the problem – and the problem (or feature, depending on your view) is not isolated to processes called from within the DirectTalk/6000 application – it affects all processes that use the SNA subsystem to communicate with a remote host.

When a process in AIX uses SNA for any form of communication, the process group to which the process belongs is sent a *SIGUSR1* (signal number 30) by the SNA subsystem when SNA is unable to carry out the request.

To understand the complications of this scenario, let's take a look at the process group concept. Each process in the system is a member of a process group. Each process group is identified by a process group ID (or PGID). By grouping processes in process groups, the system is able to send signals to all processes in the process groups. Ideally, processes in the same process group are related in some way or another.

A new process joins the process group of its parent. Because my process was called from the DirectTalk/6000 application, when the process attempts to connect to the remote host via SNA, the SNA subsystem sends a signal 30 (*SIGUSR1*) not only to my process but to all members of the process group, including some of the core DirectTalk/6000 processes. The quick fix from my application's point of view is to capture the *SIGUSR1* being sent from the SNA subsystem. This protects my process, but does nothing about other processes in the process group, which still have to be coded to trap the signal as well.

If you are writing an application that uses SNA connectivity to a remote host, it's recommended that processes that make up your application and communicate with the SNA subsystem be contained in their own process group.

Below are a few sample lines of code written to help you understand how PGIDs are inherited, and how they are assigned. (Note the use of the continuation character, '➤', which indicates that one line of code maps to more than one line of print.)

PGID.C

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

main()
{
  int a;
  printf("Original parent starting...\t(%d) (%d)\n", getpid(),
  ➤  getpgrp());
  if (fork() > 0 ) exit(0);

  printf("I am the new parent...\t\t(%d) (%d)\n", getpid(), getpgrp());
  sleep(60);
  exit(1);

  sigignore(SIGUSR1);
  a=fork();
  if (a==0)
  {
    printf("I am the first child...\t\t(%d) (%d)\n", getpid(),
    ➤  getpgrp());
    sleep(60);
    exit(1);
  }

  a=fork();
  if (a==0)
  {
    setpgid();
    printf("I am the second child...\t(%d) (%d)\n", getpid(),
    ➤  getpgrp());
    sleep(60);
    exit(1);
  }
}
```

SAMPLE OUTPUT

```
   Original parent starting...   (46706) (46706)
   I am the new parent...        (49084) (46706)
   I am the first child...       (48760) (46706)
   I am the second child...      (47478) (47478)
```

In the above sample output, the first column contains the PIDs of the original process and the children they spawned. The second column contains the process group ID of each process. The PID and PGID of the 'Original parent' process are the same. Each child spawned by the parent process is then allocated to the same process group as the parent.

In this example, if the 'first child' is used to communicate via SNA with a remote host, the SNA subsystem sends a *SIGUSR1* to the 'Original parent', 'new parent', and the 'first child' (all processes within the '46706' process group); however, if the 'second child' is used for SNA communication, the *SIGUSR1* is sent only to processes within the process group '47478', in this case affecting only the 'second child'.

Jarrod Brown
AIX Systems Programmer/Administrator (USA)

Implementing a taskbar widget

Figure 1 shows a taskbar widget that displays the progress of a task from start to completion. The implementation is achieved via a program called *taskbar.c*, and may be called by any other program to show the progress of a task.

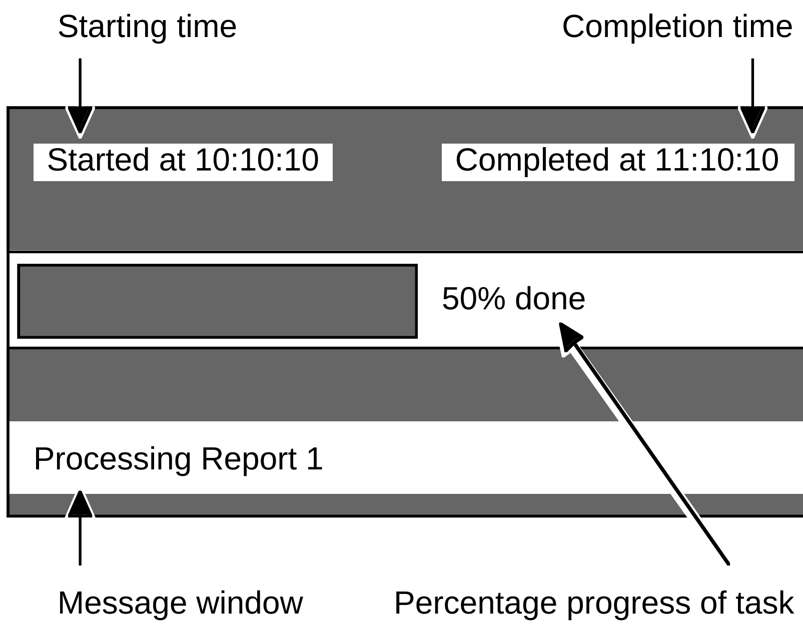

Figure 1: The widget's UI

INTERFACE FUNCTIONS IN TASKBAR.C

The following four functions comprise *taskbar.c*'s interface:

- *short MakeTaskbarWidget(void)*
- *short UpdateTaskBar(short percentage)*
- *short DisplayMessage(char *message)*
- *short EndTaskbarWidget(void)*.

STEPS TO MAKING IT WORK

1. Decide on the task whose progress is to be shown.
2. Work out the percentage completion points (the points at which the program should display 50% done, etc) for the given task.
3. Call *MakeTaskbarWidget()* once at the start of the main program.
4. Call *UpdateTaskbar()* and *DisplayMessage()* as required.

5 Compile and link *taskbar.c* to the main program as follows:

```
cc -o task task.c taskbar.c /usr/lib/libcurses.a
```

where *task.c* is the program that is to show the progress of a task.

NOTES

1 Each cursor position on the taskbar represents 2% of a task, and the taskbar, therefore, has fifty cursor positions.

2 When calling *UpdateTaskbar()*, the percentage completed must be a multiple of two.

TASKBAR.C

```
/*********************************************************************
*
*  Name      : taskbar.c
*
*  Overview  : The program creates a taskbar widget on the screen to
*              show the progress of any task from start to completion.
*
*  Notes     : 1. The following interface functions are provided by
*                 the program:
*                 - MakeTaskbarWidget   ()
*                 - UpdateTaskbar       ()
*                 - DisplayMessage      ()
*                 - RemoveTaskbarWidget ()
*
*  History   :
*  Date      Author   Description
*  -------------------------------------------------------------------
*  18/09/98  A Zaman  Initial  Build
*********************************************************************/

/*********************************************************************
*                       INCLUDE FILES
*********************************************************************/
#include  <stdio.h>
#include  <curses.h>
#include  <unistd.h>
#include  <time.h>
#include  <fcntl.h>

/*********************************************************************
*                       FUNCTION PROTOTYPES
*********************************************************************/
```

```
short  MakeTaskbarWidget       ( void );
short  UpdateTaskbar           (short percent );
short  DisplayMessage          (char *msg );
short  RemoveTaskbarWidget     (void );
short  GetTime                 (char *time );
void   DisplayCompletionTime   (void);

/*********************************************************************
*                       MODULE CONSTANT
*********************************************************************/
#define TRUE            1
#define FALSE           0
#define DONE            3

#define SUCCESS         1
#define FAILURE         0
#define UNIX_SUCCESS    0
#define UNIX_FAILURE    1

#define WINXCOR        10      /* details of primary window */
#define WINYCOR         5      /* in absolute coordinates */
#define WINHEIGHT      15
#define WINWIDTH       50

#define TBXCOR         10      /* details of subwindow for */
#define TBYCOR         11      /* displaying the taskbar */
#define TBHEIGHT        3
#define TBWIDTH        50

#define MWXCOR         10      /* details of subwindow for */
#define MWYCOR         18      /* displaying a message */
#define MWHEIGHT        1
#define MWWIDTH        50

#define CW1XCOR        11      /* details of subwindow for */
#define CW1YCOR         7      /* displaying the start time */
#define CW1HEIGHT       1
#define CW1WIDTH       19

#define CW2XCOR        38      /* details of subwindow for */
#define CW2YCOR         7      /* displaying the completion time */
#define CW2HEIGHT       1
#define CW2WIDTH       21

#define HWXCOR         26      /* details of subwindow for */
#define HWYCOR          5      /* displaying the heading */
#define HWHEIGHT        1
#define HWWIDTH        15

/*********************************************************************
```

```
*                       GLOBAL VARIABLES
*******************************************************************/

WINDOW  *wptr;      /* pointer to the main window structure */
WINDOW  *tbptr;     /* pointer to taskbar window structure */
WINDOW  *mwptr;     /* pointer to message window structure */
WINDOW  *cw1ptr;    /* pointer to staring clock window structure */
WINDOW  *cw2ptr;    /* pointer to ending clock window structure */
WINDOW  *hwptr;     /* pointer to heading window structure */

/*******************************************************************
*
*   Name        : MakeTaskbarWidget
*
*   Overview    : The function creates the taskbar widget and its
*                 associated components.
*
*   Returns     : SUCCESS, FAILURE
*
*   Notes       : 1. All window coordinates are held in symbolic
*                    constants.
*
*                 2. The following components are also created by
*                    this function:
*                    - heading window
*                    - start time display window
*                    - message display window
*
*******************************************************************/
short MakeTaskbarWidget ( )
{
  int   i;
  char  msg[40];
  char  time_now[10];

  /* initialize the screen */
  initscr ( );

  /* create the main window */
  wptr = newwin (WINHEIGHT, WINWIDTH, WINYCOR, WINXCOR);

  if (wptr == (WINDOW * ) NULL )
  {
    printf ("%s:%d:ERROR:Failed to create the window\n", __FILE__,
    ➤  __LINE__);
    return FAILURE;
  }

  /* reverse the video for the whole window */
  wattron  wptr, A_REVERSE);
```

```
for (i = 0; i < (WINHEIGHT * WINWIDTH); i ++)
     waddstr (wptr, " ");
wrefresh (wptr);

/* make the subwindow and display heading */
wattroff (wptr, A_REVERSE);

hwptr = subwin (wptr, HWHEIGHT,HWWIDTH,HWYCOR,HWXCOR);
for (i = 0; i < (HWHEIGHT * HWWIDTH); i++)
     waddstr (hwptr, " ");

wmove (hwptr, 0, 0);
waddstr (hwptr, "Task Bar Widget");
wrefresh (hwptr);

/* make the subwindow for taskbar */
wattroff (wptr, A_REVERSE);

tbptr = subwin (wptr, TBHEIGHT, TBWIDTH, TBYCOR, TBXCOR);
for (i = 0; i < (TBHEIGHT * TBWIDTH); i++)
     waddstr (tbptr, " ");

wrefresh (tbptr);

/* make the subwindow for message */
wattroff (wptr, A_REVERSE);

mwptr = subwin (wptr, MWHEIGHT, MWWIDTH, MWYCOR, MWXCOR);
for (i = 0; i < (MWHEIGHT * MWWIDTH); i++)
     waddstr (mwptr, " ");

wrefresh (mwptr);

/* make the subwindow for displaying starting time */
wattroff (wptr, A_REVERSE);

cw1ptr = subwin (wptr, CW1HEIGHT, CW1WIDTH, CW1YCOR, CW1XCOR);
for (i = 0; i < (CW1HEIGHT * CW1WIDTH); i++)
     waddstr (cw1ptr, " ");

wrefresh (cw1ptr);

/* display start time */
strcpy (msg, "Started at ");
GetTime (time_now);
strcat (msg, time_now);
wmove (cw1ptr, 0, 0);
wattroff (cw1ptr, A_REVERSE);
waddstr (cw1ptr, msg);
```

```
  /* update the screen */
  wrefresh (cw1ptr);
}

/**********************************************************************
*
*   Name        : UpdateTaskbar
*
*   Input       : Percentage (short )
*
*   Returns     : SUCCESS
*
*   Description : The function updates the taskbar to reflect the
*                 percentange of the task that's completed.
*
*   Notes       : 1. For percentages less than 2% or greater than
*                    100%, the function does nothing.
*
*                 2. For 100%, it displays the completion time in
*                    addition to updating the taskbar.
*
**********************************************************************/
short  UpdateTaskbar (short percent)
{

  static short  cur_xcorval ;
  static short  i;
  static char   msg[30];
  static task_completed = FALSE;

  /*
   * determine the x-coordinate of the percentage provided, assuming
   * the width of the bar is 50 spaces (representing 100%).
   */
  cur_xcorval = percent / 2 ;
  if (cur_xcorval <  1 ||  cur_xcorval > 50)
      return SUCCESS;

  if (task_completed == TRUE)
      return SUCCESS;

  /* highlight the percentage completed */
  wmove (tbptr, 1, 0);
  wattron (tbptr, A_REVERSE);
  for (i = 0; i < cur_xcorval; i ++)
       waddstr (tbptr, " ");

  /* write percentage done message */
  memset (msg, '\0', 30);
  sprintf (msg, "%d", percent);
  strcat (msg, "% done");
  wmove (tbptr, 1, 20);
  waddstr (tbptr, msg);

  /* update the screen */
  wrefresh (tbptr);

  /* display completion time */
  if (cur_xcorval == 50)
  {
     task_completed = TRUE;
     DisplayCompletionTime ();
  }
  return SUCCESS;
}

/*********************************************************************
*
*   Name        : DisplayMessage
*
*   Input       : Print message
*
*   Returns     : SUCCESS
*
*   Description : The function displays a given message.
*
*   Notes       : 1. The message must have fifty characters or less,
*                    otherwise it's truncated to fifty.
*
*********************************************************************/
short  DisplayMessage (char  *msg)
{
  static  char   message[51];
  static  short  len, i;

  /* copy first fifty characters of the message */
  memset (message, '\0', 51);
  strncpy (message, msg, 50);

  /* 'rightpad' the message */
  len = strlen (message);
  for (i = len; i < 50; i++)
      message[i] = ' ';

  message[i] = '\0';

  /* move the ponter to the begining of message window */
  wmove (mwptr, 0, 0);
  wattroff (mwptr, A_REVERSE);
  waddstr (mwptr, message);
  /* update the screen */
  wrefresh (mwptr);

  return SUCCESS;
}

/*******************************************************************
*
*   Name        : EndTaskbarWidget
*
*   Returns     : SUCCESS
*
*   Description : The function removes the window structure from
*                 memory.
*
*******************************************************************/
short EndTaskbarWidget (void)
{
  /* remove the window structure */
  endwin ();

  return SUCCESS;
}

/*******************************************************************
*
*   Name        : DisplayCompletionTime
*
*   Returns     : SUCCESS
*
*   Description : The function removes the window structure from
*                 memory.
*
*******************************************************************/
void DisplayCompletionTime (void)
{

  char  time_now[10];
  char  msg[40];
  short i;

  /* make window for the clock */
  wattroff (wptr, A_REVERSE);

  cw2ptr = subwin (wptr, CW2HEIGHT, CW2WIDTH, CW2YCOR, CW2XCOR);
  for (i = 0; i < (CW2HEIGHT * CW2WIDTH); i++)
       waddstr (cw2ptr, " ");

  wrefresh (cw2ptr);
  GetTime (time_now);
  strcpy (msg, "Completed at ");
  strcat (msg, time_now);
  wmove (cw2ptr, 0, 0);
  wattroff (cw2ptr, A_REVERSE);
  waddstr (cw2ptr, msg);

  /* update the screen */
   wrefresh (cw2ptr);
}

/*********************************************************************
*
*   Name        : GetTime
*
*   Input       : Address of an character array
*
*   Returns     : SUCCESS
*
*   Description : The function retrieves the current time and writes
*                 it to the address given.
*
*********************************************************************/
short GetTime (char *l_time)
{
  struct tm  *ptm;   /* pointer to time structure tm */
  long   int_time ;  /* current time in seconds returned by time() */

  time (&int_time);
  ptm = localtime (&int_time);

  sprintf (l_time, "%02d:%02d:%02d", ptm->tm_hour, ptm->tm_min,
  ➤  ptm->tm_sec);

  return SUCCESS;
}
```

TASK.C (THIS PROGRAM USES TASKBAR.C)

```
/*********************************************************************
*
*  Name      : task.c
*
*  Overview  : The program demonstrates the use of taskbar.c.
*
*  Notes     : 1. The following of taskbar.c's interface functions
*                 are called by this program:
*                 - MakeTaskbarWidget ( )
*                 - UpdateTaskbar ( )
*                  - DisplayMessage ( )
*                  - EndTaskbarWidget ( )
*
*               2. You must call EndTaskbarWidget () to re-enstate
*                  the terminal.
*  History   :
*  Date      Author   Description
*  -------------------------------------------------------------------
*  18/09/98  A Zaman  Initial  Build
*********************************************************************/

/*********************************************************************
*                       INCLUDE FILES
*********************************************************************/
#include  <stdio.h>

/*********************************************************************
*                       FUNCTION PROTOTYPES
*********************************************************************/
void   main  ( void );

/*********************************************************************
*                       MODULE CONSTANT
*********************************************************************/
#define TRUE            1
#define FALSE           0

#define SUCCESS         1
#define FAILURE         0
#define UNIX_SUCCESS    0
#define UNIX_FAILURE    1

/*********************************************************************
*                       GLOBAL VARIABLES
*********************************************************************/
/*********************************************************************
*
*   Name        : main
*
*   Returns     : SUCCESS
*
*   Description : The function displays the progress from start to
*                 completion of a specific task using taskbar widget.
*
*********************************************************************/
void main (void)
{

  /* create taskbar widget */
  MakeTaskbarWidget ( );
  /* simulate part of task with system command */
  DisplayMessage ("Starting Report 1");
  system ("sleep 5");

  /* display 1% completed on taskbar */
  UpdateTaskbar (2);
  DisplayMessage ("Report 2 completed");

  /* simulate part of task with system command */
   DisplayMessage ("Starting Report 2");
   system ("sleep 5");

  /* display 10% completed on taskbar */
  UpdateTaskbar (10);
  DisplayMessage ("Report 2 completed");

  /* simulate part of task with system command */
   DisplayMessage ("Starting Report 3");
   system ("sleep 5");

  /* display 50% completed on taskbar */
  DisplayMessage ("Report 3 completed");
  UpdateTaskbar (50);

  /* simulate part of task with system command */
   DisplayMessage ("Starting Report 4");
   system ("sleep 5");

  /* display 80% completed on taskbar */
  DisplayMessage ("Report 4 completed");
  UpdateTaskbar (80);

  /* simulate part of task with system command */
   DisplayMessage ("Starting Report 5");
   system ("sleep 5");

  /* display 100% completed on taskbar */
  DisplayMessage ("Report 5 completed");
  UpdateTaskbar (100);

  /* completed the task; remove the taskbar widget */
  EndTaskbarWidget ( );
}
```

Arif Zaman
DBA/Developer
High-Tech Software Ltd (UK)

Memory size display

The output of **lscfg** on MCA machines tells the user how much memory the queried machine has installed. However, when run on newer PCI machines, **lscfg** (even with the **-v** option) doesn't report the amount of memory.

To overcome this shortcoming, I've produced a script that queries the relevant commands/parts of the ODM database. The reader may choose either method of establishing the installed memory, as both work on both PCI and MCA machines. I have also checked it on Version 3.2.5 and 4 of AIX.

THE SCRIPT

```
#!/bin/ksh
#
# Script to determine how much memory (installed and usable) a
# machine has. It is usable on both MCA and PCI machines, with PCI
# machines also showing "good" memory as well as installed.
#
# The script calculates installed memory 2 different ways, one via
# normal commands (lsattr etc), the other by querying the ODM.
#

# Calculate the installed memory using "normal" commands

total_mem=0
lscfg | grep mem | while read a b c d
do
lsattr -El $b | grep size | read z y rest
total_mem=`expr $total_mem + $y`
done
echo "Total installed memory is "$total_mem" Mbytes"

# Calculate installed memory using the ODM database values

totalmem=0
lscfg | grep mem | while read a b rest
do
  odmget -q "name=$b AND attribute=size" CuAt
done | grep value | awk -F\" '{ print $2 }' | while read memory
do
  totalmem=`expr $totalmem + $memory`
done
```

```
echo "Total installed memory is "$totalmem" Mbytes"

#
# For a PCI machine we should also check "goodsize"
# indicating the current "good" or usable memory
#
# For an ODM version of calculating "good" memory, substitute
# attribute=goodsize (for attribute=size) in the script above
#

is_pci=`lscfg | grep -ci "PCI bus"`
if [[ is_pci -ge 1 ]]
then
  total_mem=0
  lscfg | grep mem | while read a b c d
  do
    lsattr -El $b | grep goodsize | read z y rest
    total_mem=`expr $total_mem + $y`
  done
  echo "Total usable memory is "$total_mem" Mbytes"
fi
```

Phil Pollard
AIX System Administrator (UK)

Contributing to *AIX Update*

AIX Update is primarily written by practising AIX specialists in user organizations – not journalists, consultants, or marketing people. In our view, such information is far more valuable to AIX professionals than that from other sources.

If you're interested in contributing to *AIX Update*, please download a copy of *Notes for contributors* from Xephon's Web site at *www.xephon.com*. Articles to be considered for publication can be sent the editor at *HarryLewis@compuserve.com*.

AIX news

Sterling Commerce has announced Connect:Remote Version 3.3, the company's remote management software. The software manages AIX workstations and servers, providing tools for managing software distribution and inventory, filesystems, and back-ups, and also provides a set of virus detection tools. It integrates with Tivoli TME/10 and is out now (no details on prices were received).

For further details contact:
Sterling Commerce Inc, PO Box 8000, 4600 Lakehurst Court, Dublin, OH 43017, USA
Tel: +1 614 793 7000
Fax: +1 614 793 7092
Web: http://www.stercomm.com

Sterling Software (UK) Ltd, 1 Longwalk Road, Stockley Park, Uxbridge, Middlesex UB11 1DB, UK
Tel: +44 181 867 8000
Fax: +44 181 867 8001

* * *

Hyperion has launched Integration Server, a suite of tools for building, deploying, and managing OLAP applications for data warehouses and marts, TP applications, and ERP systems. The product includes a shared OLAP metadata catalogue and supports OLAP servers from (among others) IBM and Arbor (DB2 OLAP Server and Essbase respectively); it runs on AIX, HP-UX, Solaris, OS/400, and Windows 9x/NT. Back-end databases supported include DB2, Oracle, and Sybase. It's available now and costs US$20,000 per OLAP server.

For further information contact:
Hyperion Software Corp, 900 Long Ridge Road, Stamford, CT 06902, USA
Tel: +1 203 703 3000
Fax: +1 203 595 8500
Web: *http://www.hyperion.com*

Hyperion Software, Toft Hall, Knutsford, Cheshire WA16 9PD, UK
Tel: +44 1565 633744
Fax: +44 1565 634154

* * *

IBM has announced the first of its Enterprise Storage Resource Management (ESRM) products, first previewed in June. StorWatch Reporter is for storage asset and capacity management and looks out over an IP network to discover servers attached to the network and determine how much disk filesystem capacity each server has. It runs on AIX, OS/390 Unix System Services, Solaris, HP-UX, IRIX, Windows NT, IntranetWare, and OS/2. Out now, the AIX and NT version costs US$8,000.

The company also announced Tivoli Manager for Network Connectivity, which collects alarm signals from network management applications, determines the cause of the alarm, and forwards the information to the Tivoli Enterprise Console, suppressing further alarms. It runs on AIX 4.2.1, HP-UX 10.2, Solaris 2.5 or 2.5.1, and NT 4.0 (no details on prices).

For further information contact your local IBM representative.