AIX Questions

The AIX Solution Provider Technical Support Group in Austin, Texas, supports software vendors who are developing or porting applications to AIX. This article is a compilation of questions that are frequently asked by vendors. The name of the responding Technical Support Group staff member appears after each response.

How can I pass a macro to a Fortran program?

You can pass a macro to a Fortran program by calling the C preprocessor, which in turn can pass this information to a Fortran program for compilation. The code example below contains the flags needed to preprocess macros. When `xlf` is invoked on a Fortran file with an `.F` extension, the preprocessor creates a preprocessed file that is prefixed with an `F` and the file extension is changed to a lowercase f (for example, `<filename>.F` becomes `F<filename>.f`).

The syntax of the compile is as follows:

```
xlf -d -WF,-D_AIX <filename>.F
```

where

`-d` leaves preprocessed source files produced by `cpp`, instead of deleting them. The `-WF` passes the listed options to a component that is executed during compilation. The component is `p`, `F`, `c`, `d`, `I`, `a`, or `I` corresponding to an optimizing preprocessor, the C preprocessor, the compiler, the `-S` disassembler, the Interprocedural Analysis (IPA) tool, the assembler, and the linker, respectively.

In the string following the `-W` option, use a comma as the separator and do not include any spaces. The `-D<name>` is equivalent to the `#define name`.

Note: Anything following `-WF` will be passed automatically to the `cpp`.

The following is the original Fortran program (that is, `<filename>.F`):

```
#ifdef _AIX
write (*,*) "_AIX is defined"
#endif
```

The preprocessed Fortran program (that is, `F<filename>.f`) is the following:

```
write (*,*) "_AIX is defined"
END
```

The output of the above program follows:

```
_AIX is defined
```

—Jeff Simon

Can an application written for parallel processing be run on a PowerPC?

Jeff Simon

Parallel programming requires the presence of certain Application Programming Interfaces (APIs) to enable clustering SP support. A Parallel Operating Environment (POE) for managing the development and execution of parallel applications is required to run any application that executes parallel programs on the AIX platform. A PowerPC

that is not configured to handle parallel processing will not be able to handle an application written for parallel processing. A parallel processing environment contains one or more of the following:

- IBM's Parallel Environment (PE)
- IBM's Message Passing Library (MPL)
- Message Passing Interface (MPI)
- IBM's Parallel Virtual Machine (PVM)

The flexibility of the SP system allows customers to run both parallel and non-parallel applications, but systems developed for parallel processing require that the environment contain the proper APIs and libraries.

—Jeff Simon

How can I find the hardware Ethernet address of an Ethernet adapter?

You can query your system for this information as follows:

```
lscfg -l <ethernet_device> -v | grep
Address
```

that is,

```
lscfg -l ent0 -v | grep Address
```

—Jeff Simon

Where can I find hints and tips about AIX/6000 Service?

AIX Service continues to add AIX/6000® hints and tips to the automated faxes available from the IBM Fax Information Service. This service is available free of charge to customers and field personnel 24 hours each day, seven days each week by calling the numbers listed below and following the instructions.

- Within the U.S., call 1-800-IBM-4FAX from a touch-tone phone.
- Outside the U.S., call 415-855-4FAX from your fax machine.

Figure 1 shows a few of the many faxes about AIX that are available from IBM Fax.

Wade Carlin

IBM Fax Information Service

Fax #	Description
1829	About AIX Service Hints and Tips from 4FAX
1293	Backup/Install—AIX 4.1 Installation Tips
6962	AIX Version 4.1 Quick Installation Guide
6964	AIX Version 4.2 Quick Installation Guide
3685	AIX 4.2 Installation Tips

Figure 1. AIX faxes

—Wade Carlin

What does mirroring the root volume really mean on AIX?

A simple mirroring allows users to create a copy of the default logical volumes if a disk experiences a failure during runtime.

The default root volume groups in AIX are as follows: hd1, hd2, hd3, hd4, hd5, hd6, hd8, and hd9var. All default root volume groups must be resident on the same disk. Their equivalent mirrors—those which are being mirrored—must reside on another disk. The logical volumes and the mirrored copies cannot span multiple disks.

Figure 2 shows the steps to take to extend the default root volume group from hdisk0 to hdisk1 while running as root.

```
extendvg rootvg hdisk1    (This extends root volume group to hdisk1)

chvg -Qn rootvg           (This disables QUORUM)

mklvcopy hd1 2 hdisk1     (Mirrors   /home file system)
mklvcopy hd2 2 hdisk1     (Mirrors   /usr file system)
mklvcopy hd3 2 hdisk1     (Mirrors   /tmp file system)
mklvcopy hd4 2 hdisk1     (Mirrors   / (root) file system)
mklvcopy hd5 2 hdisk1     (Mirrors   blv, boot logical volume)
mklvcopy hd6 2 hdisk1     (Mirrors   paging space)
mklvcopy hd8 2 hdisk1     (Mirrors   file system log)
mklvcopy hd9var 2 hdisk1  (/var file system)

syncvg -v rootvg

bosboot -a

bootlist -m normal hdisk0 hdisk1
```

Figure 2. Extending the default root volume group

Then you must shut down your system for the mirroring to take effect.

—Wade Carlin

How can I query the port status?

If you are writing a client/server application and errors occur when the client attempts to access the server port, you should isolate whether the application is failing and verify that the AIX (server) port is functioning properly.

There are several ways to monitor the accessibility and status of your port. Monitoring the port status can be accomplished at a high level by using the following command:

```
netstat -a | grep 50 | grep LISTEN
```

This checks to see if the port is in the "listen" mode (you can also check for other things. The `rpcinfo -p` command will report the status of the server. For example, this command reports whether the server is ready and waiting, or not available (see InfoExplorer™ for further details).

On a lower level coding approach, AIX uses the `ioctl` subroutines `getsockopt` and `setsockopt`, both documented in InfoExplorer. With these routines you can query information about existing sockets or assign new information to unused sockets.

Another lower level approach is to look at the `/usr/include/sys/socket.h`. Here you will note that the `sa_family` will contain the client address that has connected to the port, shown in Figure 3. Prior to connecting the port, the port is in the listen mode.

The process is as follows: After a connection-oriented server executes a `listen` system call in its server or test code, an

```
/*
* Structure used by kernel to store most
* addresses.
*/
struct sockaddr {
 u_char sa_len;      /* total length */
 u_char sa_family;   /* address family */
 char sa_data[14];   /* actually longer; address value */
};
```

Figure 3. Lower level coding approach

```
#include
#include
int accept (int sockfd, struct sockaddr *peer, int *addrlen)
```

Figure 4. Accept system call

actual connection from some client process is waited for by having the server execute the `accept` system call.

The `accept` system call takes the first connection request on the queue and creates another socket with the same properties as `sockfd`. The peer and `addrlen` arg's (see Figure 4) return the address of the connected peer process (the client). The new socket descriptor returned by `accept` refers to a complete association:

```
{protocol, local-addr, local-process,
foreign-addr, foreign-process}
```

where the foreign address (the client who made the connection) has been established. This allows the server to identify that a connection has been made and to log the data appropriately.

The AIX port layout may also be useful. On the Internet, 1 to 1023 are reserve ports and 1024 to 5000 are ports automatically assigned by the system. To request a port number, use one between 512 and 1023 allocated by `rresvport (int *aport)`. See Figure 5 for the port layout.

AIX Ports

Range	Usage
1-255	Well-known, privileged Internet services
256-511	Considered reserved, but not currently used by any standard Internet applications and not allocated by `rreservport()`
512-1023	Ports allocated by `rreservport()` (privileged)
1024-5000	Ports automatically assigned by system
5001-16383	Reserved for servers (not necessarily privileged)
16384-65535	Free for application use

Figure 5. AIX port layout

—David McCloud

Does AIX have a NFS Version 3 offering?

The Network File System (NFS) has been updated in AIX 4.2.1 to include support for the latest NFS protocol update—NFS Version 3. AIX NFS continues to provide distributed file system access for AIX as in the past. The AIX 4.2.1 implementation provides an NFS 2.0 client and server; therefore, it is backward compatible with the existing installed base of NFS clients and servers. Some NFS features include the following:

- The NFS client can request an asynchronous write and commit sequence for writing file data. This feature enables faster file writes to the NFS server. With the current NFS 2.0 protocol, the NFS server must write file data to disk before responding to the NFS client. If the NFS 3.0 asynchronous write request is used, this is not required.
- NFS 2.0 limited the size of `READ` and `WRITE` requests to 8 KB. The NFS 3.0 protocol relaxes the transfer size for `READ`s and `WRITE`s. The AIX implementation, like most in the industry, offers a 32 KB `READ` and `WRITE` size for both client and server. For example, with NFS 2.0 the reading of a 128 KB file would require the NFS client to send 16 individual remote procedure calls to the NFS server. With NFS 3.0, the same file could be read with four remote procedure calls.

- The NFS 3.0 protocol can now access files greater than 2 GB in size. The AIX NFS client and server, therefore, provides access to files greater than 2 GB.

Together with NFS 3.0 support, AIX 4.2.1 supports other functional changes in the NFS client and server.

NFS over TCP. The NFS client and server can use the TCP network transport for communication. Prior to AIX 4.2.1, NFS was limited to the User Datagram Protocol (UDP) transport for remote procedure calls. The default transport for AIX 4.2.1 NFS is UDP. For those environments that could benefit from TCP transport usage, the transport can be selected when the file system is mounted on the NFS client. Some examples of environments that may benefit from NFS over TCP usage would be networks with several intermediate gateways or routers, wide area networks, or environments with heavily loaded NFS servers. In all cases, NFS over TCP should provide a well-balanced network load.

Multithreaded NFS server. The NFS client and server daemons were implemented in AIX 4.2.1 using AIX's multithreading support. The NFS server daemon, `nfsd`, has been a multiprocess implementation in the past. With the new multithreaded NFS server, load balancing the server becomes much easier. NFS server threads are created and destroyed on demand as the incoming NFS client requests increase and decrease. The NFS client also takes advantage of the multithreaded approach to provide a well-balanced resource approach to reading and writing files.

Improved NFS file locking implementation. In the previous AIX releases, the NFS daemon requested by services network file locking has been a separate, user-level process. In AIX 4.2.1, the NFS file locking requests are serviced very similar to the classic NFS requests. The `rpc.lockd` daemon, a multithreaded kernel-level implementation, allows better throughput and response time. The AIX 4.2.1 NFS implementation, in certain circumstances, will provide better NFS performance than previous releases. Improvements may be observed in those environments where throughput is not limited because of CPU, network bandwidth, or disk bandwidth.

- The NFS 3.0 implementation can provide better throughput for `READ` because of the larger transfer sizes. Up to 20% more throughput can be obtained using a 32 KB `READ` size over NFS 3.0 compared to the 8 KB `READ` size over NFS 2.0.
- Using the asynchronous write capability of NFS 3.0, along with the larger `WRITE` transfer sizes, can result in an increase of two to three times the current sequential write throughput measured with NFS 2.0.
- The NFS 2.0 server implementation can provide up to 10% improvement for overall throughput or NFS operations per second.

—David McCloud

PS Command Values

Command	Description
`CPU`	The sum of the CPU usage by each of the threads
`PRI`	The highest priority of all the threads; for example, if a process has three threads with priorities 20, 30, and 40, the highest priority, 20, is displayed
`WCHQN`	1. Blank: None of the threads are waiting on a channel or none of the threads are waiting for an event 2. An address: (for example, 19e174c). Only one thread is waiting on a channel 3. An asterisk (*): More than one thread is waiting on a channel

Figure 6. The ps command values

How should the C, PRI, WCHAN on a ps command be interpreted for a multithreaded process?

For a command `ps -elUx | pg`, the values for `C` (CPU usage), `PRI` (priority), and `WCHAN` (wait channel) for a multithreaded process can be interpreted as shown in Figure 6.

—Asma Saudagar

Compiled by Jeff Simon, IBM Corporation, 11400 Burnet Road, Austin, TX 78758.