IBM'S MAGAZINE FOR AIX DEVELOPERS MARCH 1998

AIXpert

FOCUS ON INTERNET TECHNOLOGIES

Internet Technologies on AIX

IP Security

Java Servlet

AIX Client/Server GUI

Inside this Issue...

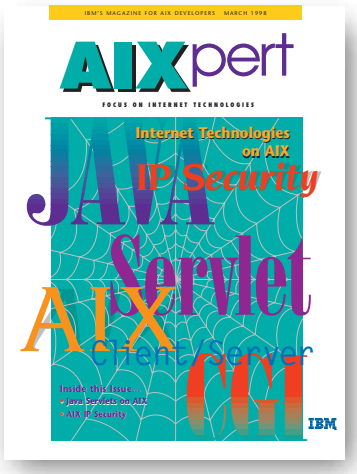
- Java Servlets on AIX
- AIX IP Security

AIXpert BACK TO CONTENTS

IBM

AIXpert DIGITAL VERSION IBM

TABLE OF CONTENTS



Click here to view a full-size version of this issue's cover.

Click here to go to a complete version of this issue for printing.

MARCH
1998

AIXpert BACK TO MAIN MENU

Move to Maui

COMMENTARY

A recent television commercial about using the Internet for business closes with the quip, "If you wanna surf, move to Maui!" to emphasize that the Internet is growing up fast. It is not the private domain of programmers and educators, nor is it exclusively a vast repository of information. The Internet is also the medium for conducting serious business through product information and public image Web sites, online stores, brokerages and travel agencies, and product support through chat sessions and forums to name a few applications. But we don't need to belabor that point. You are the people who are determining the direction and speed of Internet development with your creativity. Instead, we've put together an issue to help you by highlighting some (certainly not all) of the ways you can use AIX® to build new Internet solutions.

If you're not aware of Java™ as a basic Internet technology, you haven't been reading this magazine for long! We include two Java articles in this issue. An article on Java servlets provides an introduction to this hot new way to expand the capabilities of your web server software—a must read. This is coupled with an article that explains how to implement the client/server model using Java—a topic that has not been well documented in today's literature. Follow that with an article that explains Virtual Private Networks (VPNs) and how to use AIX IP security to implement a VPN solution on the Internet with easy-to-use, policy-based management. Capping off our Internet coverage in this issue are articles about how to minimize CGI script usage, plus a description of the new Network Station S/1000 to expand your arsenal of effective networking hardware to incorporate in your solutions.

But we're not myopic. We ve included local area network coverage in this issue as well with an article about Novell Network Services 4.1 for AIX—a solution for sharing resources across disparate operating systems and hardware. In addition, the High Performance I/O article discusses the details of using AIX's direct I/O feature to speed up certain types of I/O operations, and we have the third installment of the parallel programming on the SP2 series of articles. The popular Question and Answer column rounds out the issue.

So, move to Maui if you want. Once there, you can develop programs using AIX and its supported, open standards, knowing that you have all you need for growing your customers' electronic business.

George Noren

***George Noren**, IBM Corporation, Internal Zip 1034, 11400 Burnet Road, Austin, TX 78758. Internet: geo@austin.ibm.com. Since joining IBM in September 1979, Mr. Noren has written hardware and software manuals, including AIX and RS/6000 manuals, and was a member of the InfoExplorer™ design team. He has worked as a system administrator for several AIX systems and is a Certified AIX System Administrator. He is currently Editor in Chief of the World Wide Web site for IBM's Solution Developer Program (www.developer.ibm.com) in addition to his work with* AIXpert *Magazine. Mr. Noren studied engineering at Illinois Institute of Technology, holds a BA in English from the University of Minnesota and an MBA from St. Edwards University in Austin.*

George Noren

Using Java Servlets on AIX

By Jeff Jilg, Greg Flurry, and Michael C. Tulkoff

What are Java servlets and how can you use them? What software do you need installed to use servlets and what are some typical configurations? This introductory article answers these questions and others. It includes programming samples that you can easily deploy on AIX within minutes.

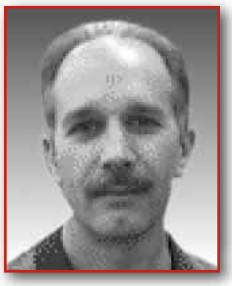
The rising popularity of Java™ has been primarily a client-side phenomenon. Java applets abound in nearly every application type. For server-side programming, Java servlets are easy to code, platform independent, and show relatively good performance.

Java servlets are server-side Java programs that are designed to extend the behavior of the server on which they reside. Servlets are typically, but not exclusively, used to extend Web servers.

Java servlets are similar to Java applets in the way they extend the functionality of Java-enabled Web servers. Since servlets perform server-side processing, they do not have a graphical user interface (GUI). For example, a Java servlet can be deployed on a Java-enabled Web server to receive data from an HTML form and update a database associated with the new information, as shown in Figure 1. In general, servlets are used as a platform-independent replacement for Common Gateway Interface (CGI) programs. CGI replacement servlets are a good alternative because they are easier to write than CGI bins. Also, there is some evidence that servlets perform better than CGI bins. Servlets provide programmers with all the resources that are available in Java, coupled with a mechanism for processing and responding to client requests.

Servlets are not strictly for Web programmers. Sun® designed the servlet model to run in conjunction with a Web server. But it is also possible to leverage the Web server and Java servlet infrastructure to create server-side programs that run in the background. For example, a servlet could be designed to monitor and record transaction types and characteristics on a commerce server. Another servlet might be designed to receive performance alerts from system management software and notify system administrators of possible impending server failure during an overload. The inherited Java methods built into servlets are designed for servlets to receive data and possibly send a response back to the original client.

Now that you understand where servlets can be deployed, you will need more information to create and use them on AIX®. The next section provides a brief overview of the servlet application programming interface (API). Then, we discuss the software you can use on AIX to develop and run servlets, followed by several servlet examples. We included a trivial 10-line example to demonstrate how simple servlets can be created, and a more extensive example that you might consider emulating for real-world



Jeff Jilg

Greg Flurry

Michael C. Tulkoff

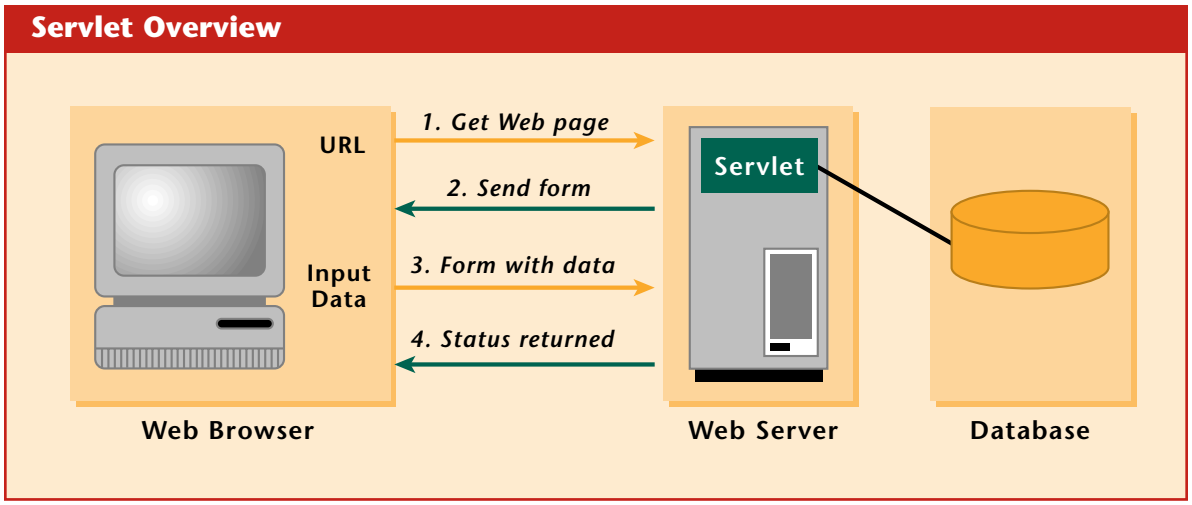


Figure 1. Overview of a Java servlet

programming. The example section is followed by a discussion of servlet programming considerations. We included some hints to help your productivity on AIX and a discussion of some IBM products and programs that exploit Java server-side logic.

Software Requirements and Configuration on AIX

Java servlets typically run from a servlet-enabled Web server. AIX has multiple viable options for the execution environment, which will be discussed below. To create a servlet, you also need a Java Development Kit (JDK) to compile the Java source code that makes up the servlet. Since Java is platform independent, any Java-compatible JDK can be used for development and compilation (that is, AIX, Solaris®, Windows NT™, etc.). The AIX JDK is available on the AIX 4.1 or 4.2 Bonus Packs, the AIX 4.3 Base, or from the IBM Web site (you can download the newest copy from *http://www.ibm.com/java*).

The respective owners of the Web server developer (for example, IBM, Netscape™) must enable Java servlets in that Web server. JavaSoft™ currently provides the enablement kit free of charge. Many servers have been enabled on AIX, including Lotus Domino™ Go, IBM Internet Connection Server, Netscape FastTrack Server™, Netscape Enterprise™, and Apache. Since enablement is fairly easy, others may follow. If your favorite Web server is not listed here, you might check its current status on the Web.

Executing a Java servlet through a Web server usually requires minor reconfiguration, which is discussed below. You should be aware of an important caveat for servlet execution. Some Web servers require the server to be shut down and restarted for the software to recognize servlets, or even new versions of servlets. For example, adding a new servlet to your servlet repository requires that you shut down the Web server and restart the FastTrack Server. You must also shut down and restart the server if you modify and recompile the servlet. *Note:* You can get around this problem by using ServletExpress from IBM, which can fix the problem for the servers supported by ServletExpress.

It is usually not necessary to test multiple Web servers to verify equivalent behavior between different Web servers. Servlets differ from applets in that they are not downloaded to a Web browser. Instead, they run directly on a server. However, if you do run multiple Web servers in your company and you intend to use one machine to do your testing, then you can run two or more servers simultaneously. Just set each one to listen to a different port (for example, set Domino Go to port 80 and FastTrack to port 81).

Depending on the architecture of the Web server, servlets are started either when the Web server starts or when they receive a request. The systems administrator can usually choose to have the servlets started within the Web server process or as a separate process. For example, the Lotus Go™ Web server allows users to modify a configuration file called `servlet.conf` to specify this behavior. It is best to have servlets run as part of the Web server (or servlet server) because it will be much faster—one of the servlets' main advantages over CGI. Much more overhead would be required to use CGI to fire up a Java program compared to using a servlet. This is because CGI would start a unique Java Virtual Machine (JVM) for each Java request. It would be much faster to just redirect the request to a running thread inside the server process.

Domino Go Web Server

The Lotus Go server currently available on the Bonus Pack provides a nearly complete package for servlet development and execution. You also will need a Java JDK for compilation and the AIX Java JDK runtime classes for execution. This packaging allows you to update the Java JDK with the newest version as updates from IBM are available (Web or Bonus Pack).

Lotus Go 4.6.1 includes the Java Just in Time (JIT) compiler-enabled JDK Version 1.1.2. It is fairly easy to configure, requiring only minor changes to the `/etc/httpd.conf` and `/etc/servlet.conf` files, which can be done manually or through the administration screen. If you misplace your Lotus Go password (as we did), the command-line version can be used (`<server_root>/cgi-bin/htadm-adduser`).

By default, servlets can be installed in the directory `/usr/lpp/internet/server_root/servlets/public`. We tested our servlets on AIX 4.2.1 with Lotus Go Versions 4.6.0 and 4.6.1 and found performance to be satisfactory. We found a minor defect in the large `OrderEntry` example servlet, which is discussed further in that section. We attribute the defect to an old (early) version of the Java Servlet Development Kit (JSDK) and found that it occurred in Lotus Go Version 4.6.0, but not in Version 4.6.1.

Netscape FastTrack Server

Enabling servlets is easy on Netscape FastTrack Server Versions 2 and 3.01 and Netscape Enterprise Server Version 2. We tested this enablement on FastTrack 3.01 on AIX 4.2.1. FastTrack currently uses Netscape's own JVM, but you must obtain the JavaSoft JSDK from the Web. As this article went to press, Netscape was considering the use of platform-native JVMs. After installing and configuring FastTrack, the JSDK was downloaded and untarred.

We followed the steps in the JSDK `README.netscape` file as summarized below.

- The `classes.zip` file was unzipped into the `<nshome>/plugins/java/localclasses` directory (where `<nshome>` is the directory that you set up as the FastTrack root directory).
- Java was enabled in the FastTrack administrative interface.
- A few lines were modified in the `<nshome>/https<host>/config/obj.conf` configuration file.
- The configuration changes were activated in the FastTrack administration screen.
- The Web server was restarted to activate the configuration changes.

The JSDK file downloaded from JavaSoft works well on AIX, although it is listed as supporting Solaris. Pure Java classes generally work fine on AIX's Java-compatible Java port. Currently, Netscape uses its own JVM port at the Java 1.02 level. Netscape has stated its intent to support servlets directly in future versions of both Enterprise and FastTrack (see "Notes for Java Programmers" at *http://www.developer.netscape.com/find/index.html*). We hypothesize that they will incorporate and package the corresponding pieces of the JSDK into their servers.

While our Netscape servlet tests were done on FastTrack 3.01, servlets also work on FastTrack Version 2.0 and Enterprise Server 2.0. Although the JSDK `README` file was easy to follow, you also might find the article "Java Servlets in Netscape Enterprise Server" about Netscape's work on servlets helpful (see *http://www.developer.netscape.com/find/index.html*).

Apache

Although we did not test Apache on AIX, servlets can be executed on Apache on AIX. Download both Apache and the JavaSoft JSDK. The JDSK `README.apache` file contains configuration and usage instructions.

ServletExpress Plug-in

ServletExpress enables the use of servlets on AIX without using Lotus Go. ServletExpress is a plug-in that allows existing Web servers to support servlets. For AIX, ServletExpress can provide servlet support in Lotus Go 4.6.1, Netscape Enterprise and FastTrack 2.01 servers, and Apache Server 1.2.4.

ServletExpress provides more than just a servlet execution environment; it also provides a graphical interface for servlet management, additional security features, and dynamic reloading of modified servlets, as well as other features. You can find additional information about ServletExpress at *http://www.ibm.com/java/servexp/*.

Servlet Programming

This section provides an overview of the basic aspects of servlet programming. The JSDK installation steps are described and the typical servlet functional flow is diagrammed. The primary servlet classes and methods are documented prior to their use later in the example section. This section also contains some important considerations concerning threaded Java servlets.

Java Servlet Development Kit

To program servlets, you must first download the Java Servlet Development Kit. The JSDK can be downloaded from the JavaSoft site at *http://jserv.javasoft.com/products/javaserver/downloads/index.html*. For AIX systems, download the Solaris version (compressed `tar` format) of the toolkit, which runs fine on AIX.

After downloading the JSDK, follow these steps:

- Decide the location for the toolkit.
- Issue a `zcat JSDK1.0.1solarisdom.tar.Z | tar xvf .`
- Add the Java classes in the toolkit to your `CLASSPATH`.
- In `ksh`, export `CLASSPATH=$CLASSPATH:<directory>/JSDK1.0.1/lib/classes.zip`.

or

- For `csh`, do a `setenv CLASSPATH $CLASSPATH:<directory>/JSDK1.0.1/lib/classes.zip`.

In addition to the class files needed to run servlets, the toolkit also contains API documentation, an executable called `srun` to run servlets stand-alone without a browser, some sample servlet source code, and the source code for the servlet classes themselves. The toolkit root directory explains this in a `README` file.

> ***A Java servlet can be deployed on a Java-enabled Web server to receive data from a Web form and update a database associated with the new information.***

Servlet Life Cycle

The servlet life cycle begins when the servlet is loaded, shown in Figure 2. As we mentioned earlier, the servlet may be loaded at server initialization time or when the first client request is received by the server. In either case, the server will load the servlet, create an instance of the implemented class, and call the servlet's `init()` method. As with any Java program, you may elect to override `init()`. Although

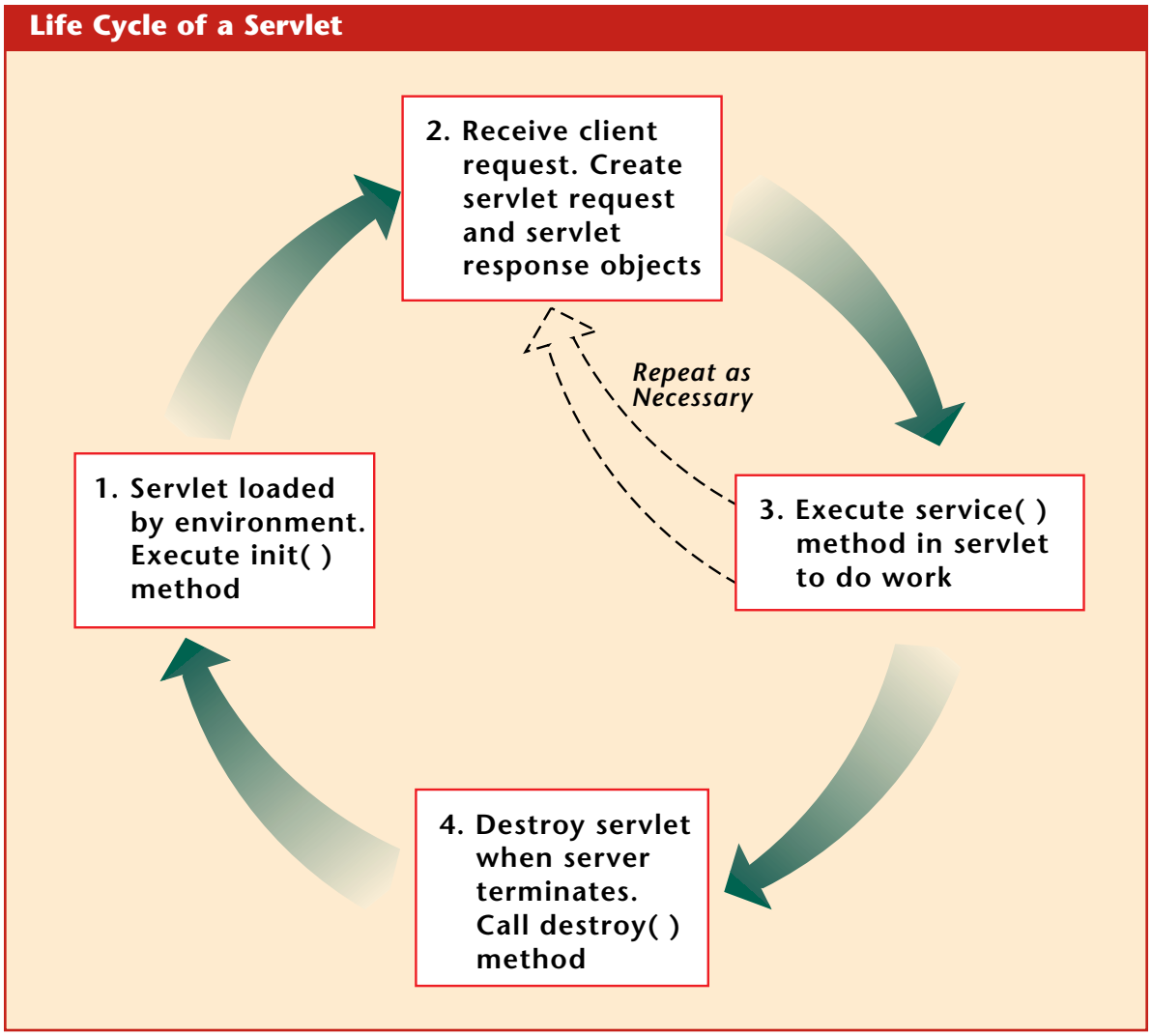


Figure 2. Servlet life cycle in servlet environment

server architectures may vary, typically the servlet will remain loaded until the server is brought down. The `init()` method is only called once per load, not every time a request is received.

The next step is the handling of a client request. Once the server has a client request, it instantiates a special object (contained in the JSDK) called a `ServletRequest` object. The server uses the `ServletRequest` object to pass information from the client to the servlet. At the same time, the server creates a `ServletResponse` object to enable the servlet to communicate back to the client.

The next and most important part of the life cycle is the server invoking the servlet's `service()` method. The `service()` method, which contains the meat of the servlet, is invoked every time a client sends a request. The `service()` method may invoke any other implemented method to process the client request. It will also use the methods in the `ServletResponse` method to interface back to the client.

Eventually, the servlet will be discarded, which typically happens when the server terminates. The servlet contains a default `destroy()` method that is invoked when this occurs. You may override the destroy method if there is any extra accounting or cleanup that you need to do—which is typically not necessary.

Servlet Classes and Methods

When coding a servlet, there are two major classes of concern: the `GenericServlet`

class from the javax.servlet package and the HttpServlet class from the javax.servlet.http class.

The HttpServlet class is actually a subclass of GenericServlet. Both classes implement the servlet interface. You would use the GenericServlet class for a servlet that communicates directly with the client program. For communicating with Web browsers, you will want to use the HttpServlet class, which allows you to create dynamic HTML without clumsy scripting languages. It contains methods that allow the servlet to respond to HTTP GET, POST, and PUT commands, just like CGI.

The GenericServlet class contains ServletRequest and ServletResponse classes. In the HttpServlet class, these are called HttpServletRequest and HttpServletResponse. In both classes, an object of each type is passed as a parameter to the service() method of the servlet. Typically, the servlet would use an InputStream to receive data from the request and an OutputStream to send data to the response. Alternatively, readers and writers could send and receive this data.

GenericServlet Class

The GenericServlet class has the following methods:

- init(): Called on the servlet load
- destroy(): Called when the server is finished with the servlet, usually upon termination
- service(): Called when a request is received; an abstract method that must be overridden and implemented if the servlet is to do any work

Figure 3 contains a shell for a generic servlet.

HttpServlet Class

The HttpServlet class also contains the init(), destroy(), and service() methods. The significant difference between the two classes is that the service() method has already been implemented in the HttpServlet class and does not need to be overridden. In fact, you will not want to override the service() method unless you are adding enhancements to the HTTP protocol or using a later version of HTTP than the one supported by the JSDK (currently it is HTTP 1.0).

In addition to these basic methods, other methods support all of the HTTP protocol commands. These methods should be overridden according to what the servlet will need to handle for a particular application. The default service method will call these HTTP-specific methods automatically when it receives the corresponding HTTP command from the client. These methods include:

- doGet(HttpServletRequest, HttpServletResponse): Handles HTTP GET, conditional GET, and HEAD requests. It returns BAD_REQUEST in default (not overridden) implementation.

```
// Howdy servlet
import java.io.*;
import javax.servlet.*;

public
class HowdyServlet extends GenericServlet {
  public void service(ServletRequest req, ServletResponse res)
       throws ServletException, IOException
  {
    ServletOutputStream out = res.getOutputStream();
    out.println("Howdy Pardner!");
  }
}
```

Figure 3. Shell for a generic servlet

- `doPost(HttpServletRequest, HttpServletResponse)`: Handles `HTTP POST` requests. It returns `BAD_REQUEST` by default.

- `doPut(HttpServletRequest, HttpServletResponse)`: Handles `HTTP PUT` requests. It returns `BAD_REQUEST` by default.

- `doDelete(HttpServletRequest, HttpServletResponse)`: Handles `HTTP DELETE` requests. It returns `BAD_REQUEST` by default.

- `doOptions(HttpServletRequest, HttpServletResponse)`: Handles `HTTP OPTIONS` requests. This method has a default implementation that determines what HTTP options are currently supported and returns a list of these options.

- `doTrace(HttpServletRequest, HttpServletResponse)`: Handles `HTTP TRACE` requests. This method has a default implementation that sends back all of the headers that were contained in the trace request.

- `string getServletInfo()`: Returns whatever string is desired to describe the servlet in case the client application has logic to display information about the servlet. By default, this method returns `NULL` and can be overridden.

HttpServletRequest Class

The `HttpServletRequest` object allows the servlet to access the HTTP client request. The servlet can access the HTTP header information and any arguments passed with the request. It contains the following methods:

- `getParameterValues(string)`: Inherited from the `ServletRequest` class. It will return the parameter values when given the name of the parameter. This method will work with any HTTP method.

- `getParameterNames()`: Inherited from the `ServletRequest` class. It will return the parameter names for the request as an enumeration of strings if they exist, or it returns an empty enumeration.

- `getQueryString()`: Can be used with the `HTTP GET` request to return the string portion of the URL for parsing.

For more complete API information, see the API documentation that ships with the JSDK.

A Word About Threads

By default, the `HttpServlet` class allows multiple threads to execute the service routine concurrently. In other words, every time a client sends a request, the server will execute the `service()` method (including the HTTP method such as `doGet`) in a unique thread. The Lotus Go server, for example, uses a thread-pooling scheme; that is, the needed threads are started beforehand and kept around in a pool. This avoids the thread-creation overhead when the server is started.

When a request is received, it is simply assigned to an available thread and returned to the pool when done. This implies that several copies of the same code can be executing at any given time (at least on a multiprocessing machine).

There are also considerations on a uniprocessor machine. Even though only one thread can be executing at any given time, many threads can be scheduled to execute and a context switch could occur at any time. For this reason, you must be careful about protecting any critical sections of your code.

Accessing a global variable is an example of a critical section. For example, in an application that keeps a running total, client one sends a request that is handled by thread one, which accesses the variable sum and increments it. The code underneath Java loaded the original value and incremented it, but a context switch happens before storing the new value. Client

two has already sent a request, so maybe thread two now tries to increment the value and is successful. Eventually, thread one will execute and complete the increment by storing the value that is no longer correct. Depending on what data is being accessed, this problem could end up a lot worse.

To get around this problem, you must first identify the critical sections, then make them "thread-safe." The easiest way to accomplish this is to synchronize the critical sections. Good Java practice would dictate that you synchronize whole methods instead of lines of code within them.

The following example shows a thread-safe `doGet`:

```
public  synchronized void doGet() {
   // stuff here
}
```

If you wish to protect your entire servlet from running in concurrent mode, you can override the behavior of the `HttpServlet` class by using the `SingleThreadModel` keyword, shown in Figure 4.

```
public class TestServlet extends HttpServlet
   implements SingleThreadModel {
} // end TestServlet
```

Figure 4. Overriding the HttpServlet class

Servlet Examples

Three ways to invoke a servlet include:

- Pointing a browser at the servlet's URL; for example, *http:myserver/servlet/HowdyServlet*
- Referencing the servlet in the `action` parameter of an `HTML <FORM>` tag; for example, `<form method=post action=/servlet/SSIServlet> ... </form>`
- Referencing the servlet in an HTML `<SERVLET>` tag in an `.shtml` file served by a Web server enabled for server-side includes (also known as server-side embeds); for example, `<servlet SSIServlet code=SSI-Servlet.class> ... </servlet>`

We demonstrate all three methods in the examples below. Any good servlet reference will contain more detail about the first two mechanisms. For additional details about server-side includes, see the following Web sites:

- *http://www.ics.raleigh.ibm.com/pub/icswp004.html*
- *http://jserv.javasoft.com:80/products/javaserver/documentation/webserver1.0.2/servlets/servlet_tutorial.html*
- *http://www.ics.raleigh.ibm.com/pub/wpgl0mst.htm.*

Compiling and Running Servlets

Once a servlet is coded, it must be compiled. Although the following information is specific to Lotus Go, the process is similar on other servers. Make sure that the `CLASSPATH` points to class libraries for both the JDK and the JSDK. Set the `CLASSPATH` environment variable or use the `classpath` flag on `javac`. For example, assume you installed JSDK separately on your system:

```
setenv CLASSPATH=/usr/lpp/internet/
  server_root/java/lib:/jsdk/lib/
  classes.zip
javac MyServlet.java
```

After compilation, you must copy `MyServlet.class` to the `<server_root/>servlets/public` directory or the directory appropriate for your Web server configuration (`/usr/lpp/internet/server_root/servlets/public` default for Lotus Go). Then you must restart your Web server.

To run a servlet via a URL, start a Web browser, such as Netscape Navigator™. Point the browser to *http://HOSTNAME:PORT/servlet/MyServlet* (where *HOSTNAME:PORT* is your server, *myserver:80*). The browser contacts the Web server that invokes the `service()` method of the servlet, which can perform a variety of actions based on programming tasks desired.

HowdyServlet via URL

The `HowdyServlet` is a trivial example (shown in Figure 5) that demonstrates the simplicity of servlet programming. Compile and run it using the information above. You should be able to access it easily by entering the following URL into a Web browser: *http://HOSTNAME:PORT/servlet/HowdyServlet*. The browser will indicate a reply "`Howdy Pardner!`"

SSIServlet via <servlet> Tag

Another method for accessing servlets is through the server-side include (SSI) protocol in Web servers. Figure 6 shows the `ssi.shtml` file used to reference the servlet, and Figure 7 lists a simple servlet that is invoked as the result of the Web server processing the `<servlet>` tag. The servlet is simple to invoke. Just enter into a Web browser *http://HOSTNAME:PORT/ssi.shtml*.

```
// Howdy servlet
import java.io.*;
import javax.servlet.*;

public
class HowdyServlet extends GenericServlet {
  public void service(ServletRequest req, ServletResponse res)
       throws ServletException, IOException
  {
    ServletOutputStream out = res.getOutputStream();
    out.println("Howdy Pardner!");
  }
}
```

Figure 5. HowdyServlet.java

```
<html><head><title>Testing SSI</title></head>
<body>
<h2>The SSI example</h2>
<servlet name=SSIServlet code=SSIServlet.class>
</servlet>
</body></html>
```

Figure 6. ssi.shtml

```
import java.io.*;

import javax.servlet.*;
//import javax.servlet.http.*;

public
class SSIServlet extends GenericServlet {
  public void service (ServletRequest req, ServletResponse res)
       throws ServletException, IOException
  {
    ServletOutputStream out = res.getOutputStream();
    out.println("Server side includes work!");
  }
}
```

Figure 7. SSIServlet.java

```
<html>
<head><title>Forms Servlet Test</title></head>
<body>
<h2>Order Selection</h2>
<p>Please select the items you wish to order, then press Submit:</p><hr>
<form action=/servlet/OrderEntryServlet method=POST>
<p>Enter name: <input name=theName></p>
<p>Department Billed:
<input type=radio name=theDept value="D23" checked>D23
<input type=radio name=theDept value="D24" >D24
<input type=radio name=theDept value="D25" >D25
<p>Select items required: <br>
<input type=checkbox name=Pen >Pen @ $5.25<br>
<input type=checkbox name=Pencil >Pencil @ $0.25<br>
<input type=checkbox name=Eraser >Eraser @ $1.15<br>
</p>
<p> <input type=submit value=Submit Order></p>
</form><hr>
</body></html>
```

Figure 8. OrderEntry.html

OrderEntry Servlet via HTML

You can access servlets from within HTML forms. The next section describes the `OrderEntry` servlet and its logic flow. The `OrderEntry.html` file (Figure 8) should be invoked via *http://HOSTNAME:PORT/OrderEntry.html*.

The Web browser will display the corresponding HTML and invoke the `doPost()` method of the `OrderEntryServlet` (overrides the Web server `POST`), shown in Figure 9. The logic flow in the following section describes the servlet processing of the form information.

While it may seem easy to put static HTML in a form on the `OrderEntry.html` text, this is a limiting mechanism. The HTML contains hard-coded entries, such as the department values D23, D24, and D25. To avoid this, use Java servlets, which can dynamically obtain this data from files or databases. The next example demonstrates this capability.

OrderEntryServlet via URL

Figure 9 shows the source code for a rudimentary order-entry servlet. The `OrderEntryServlet` extends the `HttpServlet` class in order to leverage its built-in HTTP handling framework. `OrderEntryServlet` is invoked by a URL *http://HOSTNAME:PORT/servlet/OrderEntryServlet*.

It dynamically generates an HTML form asking for customer information and product choices (see Figure 10). Once the user submits the completed form, `OrderEntryServlet` processes the information from the form and confirms the order with additional dynamically generated HTML.

A detailed explanation of this `OrderEntryServlet` logic flow is described below. See Figure 10 to follow this logic. It is also helpful to compile and run the example to see it in action.

When the browser issues an `HTTP GET`, the Web server loads `OrderEntryServlet` and calls `OrderEntryServlet.init()`. The Web server then calls `OrderEntryServlet.doGet()` and it generates the HTML form. The Web server returns the form to the browser, which displays it. Once you complete the form and press the Submit button, the browser again contacts the Web server and issues an `HTTP PUT`. The Web server then calls `OrderEntryServlet.doPost()`, and it gathers the parameters from the form, calculates the total cost of the selected products, and generates the HTML for the response. The Web server returns the response to the browser, which displays it.

```
 /*  OrderEntry.java */

import java.io.*;
import java.util.*;
import java.text.*;

import javax.servlet.*;
import javax.servlet.http.*;

public
class OrderEntryServlet extends HttpServlet {

  String prodName[];
  double prodPrice[];

  final String deptName[] = {"D23", "D24", "D25"};
  int i;
  String oName, dName;

  public void init(ServletConfig config)
 throws ServletException
  {
    final String pName[] = {"Pen", "Pencil", "Eraser"};
    final double pPrice[] = {5.25, 0.25, 1.15};

    super.init(config);

    // initialize "database"
    prodName = pName;
    prodPrice = pPrice;

    // open connection to transaction engine (not really in this example)
  }

  public void destroy() {
    // close connection to transaction engine -(not really in this example)
  }

  public void doGet (HttpServletRequest req, HttpServletResponse res)
       throws ServletException, IOException
  {
    // set up initial html
    res.setContentType("text/html");
    ServletOutputStream out = res.getOutputStream();
    out.println("<html>");
    out.println("<head><title>Forms Servlet Test</title></head>");
    out.println("<body>");
    out.println("<h2>Order Selection</h2>");
    out.println("<p>Please select the items you wish to order, then press Submit:</p><hr>");

    // set up the form - first do name
    out.println("<form action=/servlet/TestServlet method=POST>");
    out.println("<p>Enter name: <input name=theName></p>");
    // do department information
    out.println("<p>Department Billed: ");
    for (i=0; i<deptName.length; i++) {
      if(i==0) { // make the first one the default
        out.println("<input type=radio name=theDept value=" + deptName[i] + " checked>" + deptName[i])
;
      }else{
        out.println("<input type=radio name=theDept value=" + deptName[i] + ">" + deptName[i]);
      }
```

(continued on next page)

Figure 9. OrderEntry.java

(continued from previous page)

```
    }
    // do items orderable
    out.println("<p>Select items required: <br>");
    for (int i=0; i<prodName.length; i++) {
      out.println("<input type=checkbox name=" + prodName[i] + ">" + prodName[i] + "&#09&#09@ $
          " + prodPrice[i] +"<br>");
    }
    out.println("</p>");
    // do submit button
    out.println("<p><input type=submit value=Submit Order></p>");
    // end form
    out.println("</form><hr>");
    out.println("</body></html>");
  }

  public void doPost (HttpServletRequest req, HttpServletResponse res)
       throws ServletException, IOException
  {
    // set up initial part of response
    res.setContentType("text/html");
    ServletOutputStream out = res.getOutputStream();
    out.println("<html><head><title>Forms Servlet Test</title></head>");
    out.println("<body><h2>Order Results</h2><hr>");

    // Get client's form data & process
    double total = 0.0;
    Enumeration values = req.getParameterNames();
    while (values.hasMoreElements()) {
      String zname = (String)values.nextElement();
      String zvalue = req.getParameterValues(zname) [0];
      // String zvalue = req.getParameter(zname);  USE this on older JSDKs
      if (zname.equals("theName")) {oName = new String(zvalue);}
      else if (zname.equals("theDept")) {dName = new String(zvalue);}
      else for (i=0; i<prodName.length; i++) {
        if (zname.equals(prodName[i])) {
          total = total + prodPrice[i];
        }
      }
    }
    // at this point the servlet would create a transaction for the order

    // format response
    NumberFormat nf = NumberFormat.getInstance();
    nf.setMaximumFractionDigits(2);
    nf.setMinimumFractionDigits(2);
    nf.setMaximumIntegerDigits(2);
    nf.setMinimumIntegerDigits(1);
    String oTotal = nf.format(total);
    out.println("<p>Thank you for your order " + oName + ". ");
    out.println("Your department, " + dName + ", will be charged $" + oTotal + ".</p>");
    out.println("<hr></body></html>");
  }

  public String getServletInfo() {
    return "A servlet that shows HTML processing via GET and POST";
  }
}
```

Figure 9. OrderEntry.java

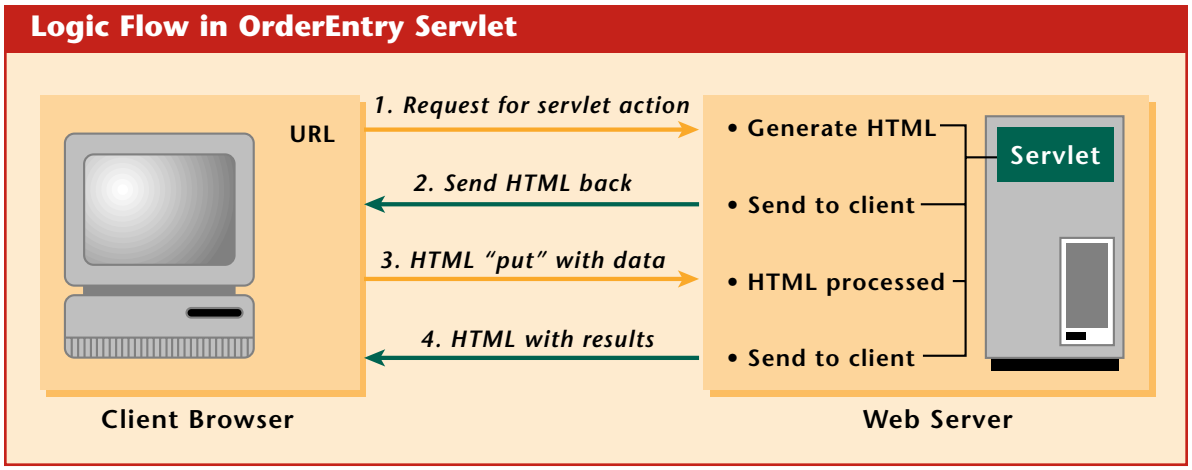


Figure 10. OrderEntry servlet logic flow

The `OrderEntryServlet.init()` method simulates opening a database and populating an array of product information. In this example, the information comes from static data; in a real-world servlet, `init()` would open a database to retrieve the product information and would also open a connection to some transaction engine on a third-tier server. The `destroy()` method in this example does nothing, because there is nothing to do; in a real-world servlet, `destroy()` would close its connection to the transaction engine.

The `doGet()` method, invoked when a browser does an `HTTP GET` operation referencing `OrderEntryServlet`, generates an HTML page containing the form for collecting order information. It does this by writing the HTML to a `ServletOutputStream`. The form calls for a `POST` response, and directs the `POST` to `OrderEntryServlet` using the HTML statement `<form action=http://HOSTNAME:PORT/servlet/OrderEntryServlet method=POST>` for our testing with Lotus Go.

The action may be somewhat different, depending on your Web server configuration requirements. The information used to construct the form statement can also be parameterized at initialization time. The form first asks for a customer name via a text input field, then a department number via radio button, product selections via a set of check boxes, and finally presents a "Submit" button.

The `doPost()` method, invoked when the user hits the Submit button, collects the order information, processes it, and reports the order results to the user. To report the results, like `doGet()`, it generates an HTML page containing the response—in this case, the total cost of the order—by writing the HTML to a `ServletOutputStream`. Then, `doPost()` first sets the total cost of the order to $0.00 and iterates through the parameters passed back via the form. It then collects the value of the user name and department, saving the returned strings.

For the products, it simply determines if a product was selected or not (if a product is not selected, the parameter name does not appear in the enumeration returned by `getParameterNames()`); when a product is selected, `doPost()` adds its price to the total. At this point, `doPost()` in a real-world servlet would create a transaction for the order, and perhaps retrieve additional information, such as the expected delivery date.

Finally, `doPost()` formats the response to the order form by simply echoing the name and department entered by the user and displaying the total cost of the items ordered; in the real world, any additional information about the transaction would also be displayed in the response.

As mentioned earlier in the article, older JSDKs have a bug that does not allow proper array processing. This bug is highlighted in the `doPost()` method on the

`String zvalue` line. We have indicated a way to get around this bug on older JSDKs when you only have a single element to extract from the array.

This simple example does not demonstrate some additional considerations for writing servlets, such as processing initialization parameters, dealing with multithreaded synchronization issues, or exception handling. For a good discussion of such issues, see "Servlet Tutorial" at *http://java.sun.com/products/jdk/1.2/docs/ext/servlet/servlet_tutorial.html*.

Real-World Servlet Deployment

The most visible deployment of servlets within IBM is in IBM's Network Computing Framework (NCF). The NCF is an open, pluggable, standards-based, multi-platform framework for creating, deploying, and managing e-business solutions across the enterprise. The elements of NCF include:

- A set of Web application servers supporting an Object Request Broker (ORB) and a JVM
- A standardized interface to data and materials available on a Web server via a Web browser/Java applet model
- A Java programming model based on applets, servlets, and JavaBeans™
- Internet-ready protocol support, such as HTTP and Internet Inter-Orb Protocol (IIOP), which links JavaBeans components
- Middleware development tools for application development and content authoring
- A set of "connector" services that provide access to existing data, applications, and external services
- A set of built-in collaboration, commerce, and content services that provide a foundation for an industry of partner-built solutions and customizable applications for e-business

In NCF, servlets provide the anchor for every application. NCF supports servlet interaction via HTTP, HTTPS, and IIOP. HTTP interactions proceed much as described above. HTTPS interactions bring in additional security using public key encryption techniques. With IIOP, an applet running on the client can establish an IIOP connection to a servlet in the Web server and pass method calls back and forth, permitting a much richer and more object-oriented interaction.

AIX is one of the primary platforms for NCF deployment. For additional information about the NCF, see *http://www.software.ibm.com/ebusiness/ncf/*.

***Jeff Jilg**, IBM Corporation, 11400 Burnet Road, Austin, TX 78758.E-mail: jjilg@austin.ibm.com. Dr. Jilg is a senior AIX system architect currently responsible for AIX Java technical strategy. His MS in Computer Science from the University of Texas at El Paso complements his PhD from Texas A&M University in the same field.*

***Greg Flurry**, IBM Corporation, 11400 Burnet Road, Austin, TX 78758. Mr. Flurry is a senior technical staff member in the RS/6000 Division. His responsibilities, as part of the Systems Architecture & Technical Strategy team, include network computing and Java. He has a BS in Electrical Engineering from Vanderbilt University and an MS in the same field from the University of Kentucky.*

***Michael C. Tulkoff**, IBM Corporation, 11400 Burnet Road, Austin, TX 78758. Mr. Tulkoff is an IBM advisory software engineer. He has worked as both an AIX developer and AIX system architect. He is currently working on Business Discovery Solutions in the Global Business Intelligence Solutions industry group, where he is heavily using Java and Web-based programming. Mr. Tulkoff holds a BS in Computer Science from the Georgia Institute of Technology.*

High Performance I/O

By Bret R. Olszewski and David B. Whitworth

The challenge of supplying high-performance I/O to applications is never ending. AIX has provided services such as I/O coalescing, read-ahead/write-behind algorithms, and software striping to maximize device throughput. These algorithms frequently trade processor cycles (CPU time) for device throughput. Further optimization, improving CPU cycles for I/O, is possible in some circumstances with the direct I/O feature of AIX 4.3.

To understand direct I/O, it is necessary to review the basics of AIX® disk I/O. Since AIX is a general-purpose operating system, it supports a variety of mechanisms for disk I/O. Figure 1 shows the component stack for three cases of disk I/O: file system, memory-mapped I/O, and raw I/O.

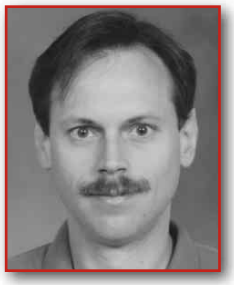
Starting from the bottom of the stacks, the layer closest to the physical disk device is the device driver. The device driver understands the characteristics of the physical device and how to make it perform primitive read, write, and status operations. On top of the device drivers is the logical volume manager (LVM). The LVM layer is rich in function. Some of its capabilities include logical volumes that span multiple physical disks, mirroring, and error recovery for physical disk problems. Above the LVM are the file systems.

AIX supports several file systems, including the journaled file system (JFS), network file system (NFS), distributed file system (DFS), CD-ROM file system (CDRFS), the special file system (SPECFS), as well as others. This article will concentrate on the JFS and the SPECFS, although many of the principals of these file systems apply to others as well. Above the file systems is the logical file system (LFS). This layer makes it possible for user applications to do I/O to different file systems using a single set of program interfaces. Each disk I/O involves a trip through the appropriate layers of software. The path length, or number of instructions required for the disk I/O, is the sum of the instructions executed in each layer. Figure 1 shows the disk I/O paths.

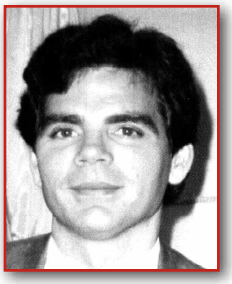
File systems typically support the following basic functions:

- **Consistency.** If a file is being read and/or written by multiple threads, each thread gets a consistent view of the data. The file system serializes access to the physical file data. This primitive consistency model is usually augmented by application locks for sophisticated applications.

- **Buffering.** Application programs typically have file access patterns that are inefficient with regard to the physical disk devices. When an application opens a file and performs reads from it, the application reads can involve disk I/O or they can be resolved with buffered data. For example, a program doing 4096 sequential one-byte read



Bret R. Olszewski



David B. Whitworth

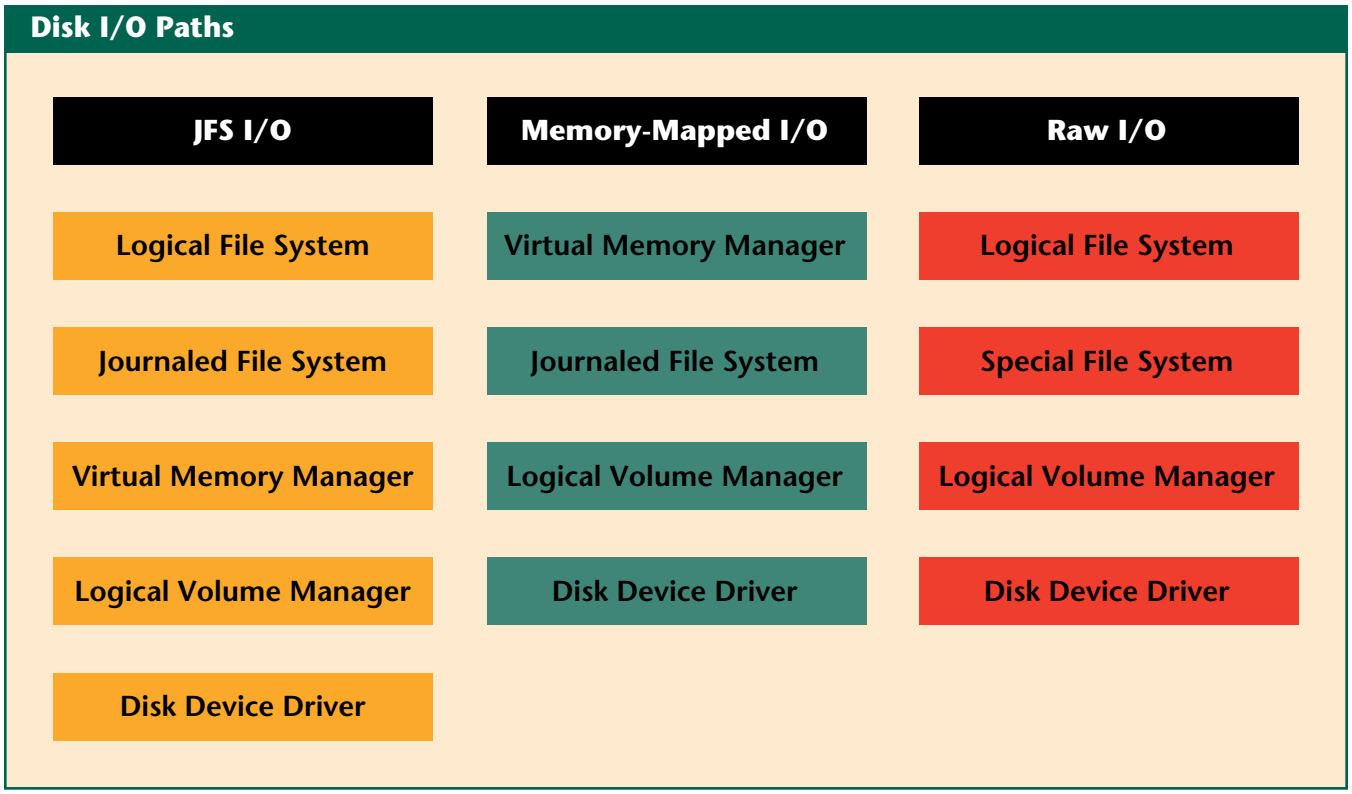


Figure 1. Disk I/O Paths

system calls will typically only need a single 4096 disk read. When data is read, a copy of the data is normally kept in memory, or buffered, as a function of AIX's virtual memory manager. If a file is frequently accessed, the file data may remain in system memory so that physical disk I/O is not needed.

- **Allocation.** A file system manages space. As files are extended, the file system allocates disk blocks. When files are truncated or deleted, the file system takes back disk blocks as free space.

- **Resource management.** A file system can manage resources such as buffer management and space quotas. It can limit how much buffering is allowed for any particular file, based on the amount of memory available as well as file usage pattern. It can limit how many disk blocks may be allocated to any user.

File system I/O via JFS is performed through system calls such as `read` and `write`. These system calls manage the file data and deal with buffering. The buffering of file data is contained within the kernel address space. This means that the file data cannot be directly accessed by user programs. Data is moved to or from user space via the system calls (that is, `read` and `write`).

Memory-mapped I/O is a means of mapping files directly into program address space (available on AIX via the `shmat` and `mmap` system calls). In the case of memory-mapped I/O, the file data is accessed by memory location (that is, not via `read/write` system calls) and no copying of data is needed. The granularity of access is a 4096 byte page. Because reads and writes are handled transparently by the virtual memory manager, applications that share files for updating must provide their own serialization, typically via mutex locks, to ensure consistency of data.

Since the underlying file system supports allocation and resource management, raw I/O bypasses the traditional file system completely. On AIX it typically passes through a thin veneer of function in the SPECFS. Essentially, it skips file system function completely, not providing any consistency, buffering, allocation, or resource management services. Because of its limited service, applications that have multiple threads updating data must provide serialization. The application also must manage allocation of space within the logical volume. Since there is no buffering, stricter alignment rules apply to data and correctly aligned buffers require no copying. Raw I/O is typically the most efficient in path length—that is why it is normally used for large database benchmarks such as TPC-C and TPC-D.

An alternative to kernel caching would be to have the file contents cached in user addressable memory. User caching has potential drawbacks. Consider what happens if a thread accidently modifies file storage, perhaps by an errant pointer reference. The accident will not be caught and the file data may be corrupted. Other hardware-related issues, such as the need to use instructions that are only legal in kernel mode to change addressability as well as address space limitations in 32-bit microprocessors, limit the effectiveness of user space caching with full UNIX® semantics. For many applications, AIX's support of user addressable caching via the `shmat` and `mmap` system calls is efficient.

Direct I/O Implementation

Most users prefer the convenience of using file systems, but some require greater performance. A side effect of the continual increase in microprocessor performance has been that memory bandwidth has not always kept up. To put it bluntly, the speed that data can be copied from one memory location to another is dependent on memory subsystem performance and is relatively independent of microprocessor speed. This manifests itself in several ways, one of which is that block copy operations become a significant percentage of the execution time for some I/Os. In fact, it is possible on some systems to achieve I/O bandwidths that are limited by the amount of system processing power—not the physical disk devices. To mitigate this effect, direct I/O merges some of the characteristics of file system I/O and raw I/O.

Programs that are typically CPU-limited and do lots of disk I/O are good candidates for direct I/O.

There is no such thing as a free lunch. Direct I/O imposes a number of restrictions on programs. Because direct I/O is uncached, as is raw I/O, no copy operation is required. The flip side of this is that there is no buffering, so two reads of the same data will result in two disk I/Os. Also, the alignment of the user buffer must be compatible with the file allocation. The alignment requirement is due to the fact that the disk data is moved into the user buffer by direct memory access (DMA) from the disk device.

Direct I/O does not benefit from VMM `read ahead` or `write behind`, so it may be necessary to increase the size of the I/Os an application does to match the sequential throughput the application gets with normal I/O. I/Os of 32 KB to 128 KB or larger should provide good throughput.

At present, direct I/O is only supported for program working storage. Direct I/O will not work to client segments (that is, files mapped to file systems such as NFS or DFS) or to shared memory segments mapped with `mmap`. Figure 2 summarizes the capabilities and restrictions of the various I/O mechanisms.

So what types of programs are good candidates for direct I/O? Programs that are typically CPU-limited and do lots of disk I/O. In particular, "technical" codes that have large sequential I/Os are good candidates. Applications that do numerous small I/Os will typically see less performance, because

Comparison of Function

I/O Mechanism	F/S R/W	Memory Mapped	Direct	Raw
Alignment required	No	Yes	Yes	Yes
Buffered	Yes	Yes	No	No
Read/write consistency	Yes	Yes	Yes	Yes
Read ahead	Yes	Yes	No	No
Write behind	Yes	Yes	No	No
Copy required	Yes	No	No	No
Serialization (`fcntl`, `flock`, `lockf`)	Yes	No	Yes	No
Client segments	Yes	NFS	No	No
Mmap segments	Yes	N/A	Yes	No
Compression	Yes	Yes	No	No
Striping	Yes	Yes	Yes	Yes
AIO support	Yes	Yes	Yes	No

Figure 2. Comparison of function

direct I/O cannot do `read ahead` or `write behind`. Applications that have benefited from striping are also good candidates.

Performance Benchmarks

To compare the amount of CPU consumed for raw I/O, direct I/O, and traditional file system I/Os, we use a set of disk performance benchmarks. The performance is measured by a program that does either sequential or random reads from varying numbers of disks. The sustained throughput and CPU utilization are reported. Our tests are performed with multiple disks and adapters. Your performance will vary, depending on system configuration.

The JFS and Direct tests read one JFS file per disk and the Raw test reads one logical volume per disk. The random tests do 4 KB I/Os. The JFS and Direct I/O sequential tests do 32 KB I/Os and the Raw sequential test does 128 KB I/Os. The Raw sequential test has a slight advantage because it does larger I/Os. The benchmark ensures that none of the data to be read is cached so that all reads come from the disks.

Results are included for the two different systems: the 12-processor Model S70 symmetric multiprocessor (SMP) and the uniprocessor E30. The two systems provide a good comparison because of the difference in the relative memory bandwidth available. Figure 3 shows the CPU utilization of the E30 system for varying levels of disk throughput doing sequential reads.

Sequential reads allow maximum disk bandwidth because disk seeks are avoided. Notice that reading files through standard JFS saturates the CPU at less than 20 megabytes per second, while direct I/O and raw I/O are consuming 15% or less of the CPU at the same throughput. Also notice that raw I/O is still more efficient than direct I/O, but they have much closer performance compared with standard JFS I/O.

Figure 4 shows the CPU utilization of the S70 system on the same benchmark. Since the memory bandwidth of the system is much higher than the E30, we were unable to saturate the CPU of the S70 with disks available for this benchmark. Notice that, even though the S70 has much more memory bandwidth, the shapes of the curves are similar to those on the E30. This indicates a

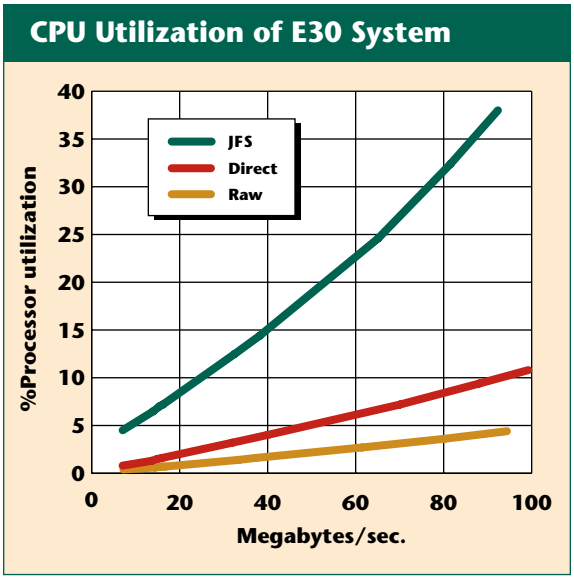


Figure 3. Disk throughput for sequential reads

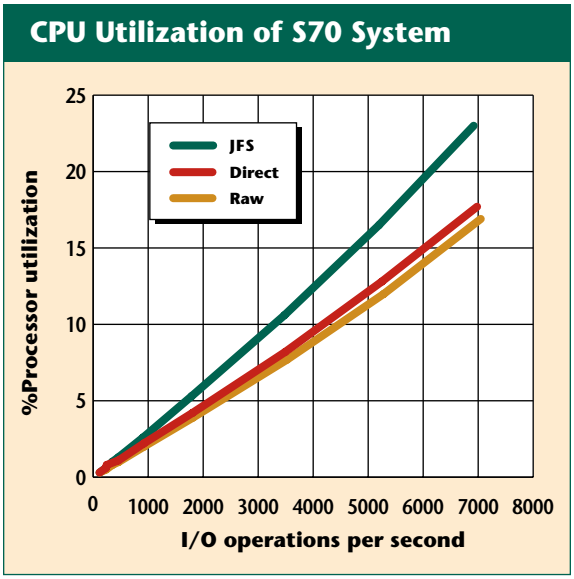


Figure 4. Disk throughput for sequential reads

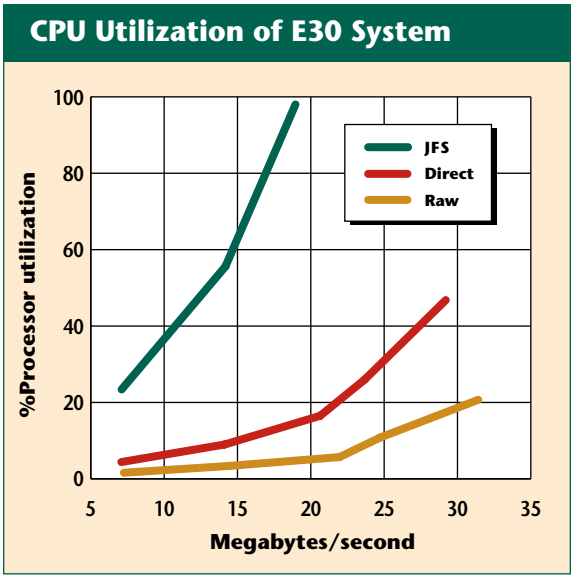


Figure 5. Random I/O

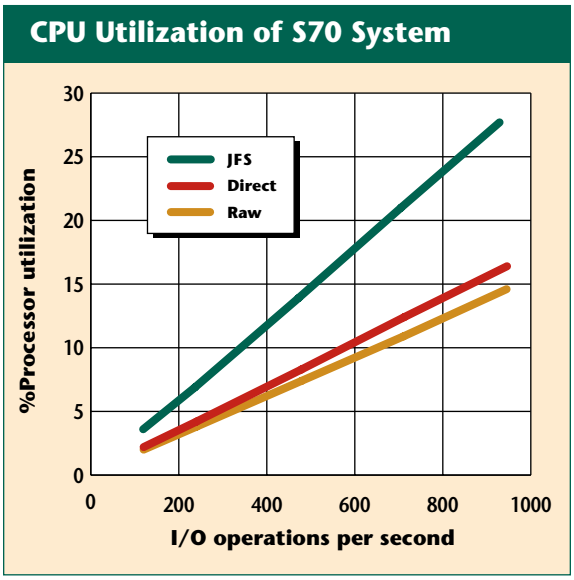


Figure 6. Random I/O

similar level of benefit of direct I/O on systems with high memory bandwidth.

Figures 5 and 6 show the CPU utilization of the E30 and S70 systems for the random I/O part of the benchmark. Since the random I/Os are smaller (4096 bytes instead of 32 kilobytes), the copy overhead of the I/O is less for the random test than for the sequential test. Put another way, a single, large I/O has efficiencies in path length compared to several small I/Os, even if the total amount of read or written (and copied in the standard JFS case) is the same. In both cases, the benefit of direct I/O is substantial, though considerably less than in the sequential cases.

Using Direct I/O

The use of direct I/O is fairly simple; none the less, we recommend consulting standard AIX documentation before modifying applications. One thing to consider is that

direct I/O is a function of JFS and is not supported on most other file systems. For example, trying to open a file mounted over NFS for direct I/O will function correctly, but not benefit from direct I/O performance. AIX provides the `ffinfo` system call to query the required alignment, minimum read/write size, and maximum read/write size of the file using direct I/O. The `ffinfo` system call only returns meaningful information when used on JFS files. Direct I/O is enabled via the `O_DIRECT` flag on the `open` system call.

Figure 7 shows a simple program that does a single 4 KB read via direct I/O. The key points are include files, the open option, buffer alignment, and the `ffinfo` call. The include file `fcntl.h` is needed for the open option. The include file `sys/finfo.h` is needed for the `finfo` system call. On the open, `O_DIRECT` is used to inform the file system that direct I/O is desired. The buffer is allocated from the heap via `malloc`. However, `malloc` does not guarantee alignment, so the program `malloc` has twice as much space as needed and uses an aligned buffer

```
#include <fcntl.h>
#include <sys/finfo.h>

main (argc, argv)
int     argc;
char    *argv[];
{
#define BUFFER_SIZE 16384
#define ALIGN 4096

        char    errmsg[80];
        char    *filename;
        int     fd;
        char    *buffer;
        struct diocapbuf buf;

        /* 4 KB align the I/O buffer */
        buffer = (char *) malloc(BUFFER_SIZE+ALIGN);
        buffer = (char *)(((int)buffer + ALIGN) & 0xfffff000);

        if ( argc < 2) {
                printf("usage: %s <data file>\n", argv[0]);
                exit ( 1 );
        }
        filename = &argv[1][0];

        if ( ( fd = open ( filename, O_DIRECT | O_RDONLY ) ) < 0 ) {
                sprintf (errmsg, "%s: cannot open %s", argv[0], filename);
                perror (errmsg);
                exit (1);
        }

        if ( ffinfo ( fd, FI_DIOCAP, &buf, sizeof(buf) ) < 0 ) {
                sprintf (errmsg, "%s: ffinfo failed %s", argv[0], filename);
                perror (errmsg);
                exit (1);
        }
        printf ( "buf.dio_offset = %d\n", buf.dio_offset );
        printf ( "buf.dio_max = %d\n", buf.dio_max );
```

(continued on next page)

Figure 7. Example of single 4 KB read via direct I/O

(continued from previous page)

```
        printf ( "buf.dio_min = %d\n", buf.dio_min );
        printf ( "buf.dio_align = %d\n", buf.dio_align );
         if (buf.dio_align != ALIGN)
              printf("alignment does not allow direct I/O!\n"):


        if ( read ( fd, buffer, BUFFER_SIZE ) < 0 ) {
                sprintf ( errmsg, "%s: bad read", argv[0]);
                perror ( errmsg );
                exit ( 1 );
        }

        close ( fd );
        exit ( 0 );
}
```

Figure 7. Example of single 4 KB read via direct I/O

within the `malloc` allocation. The `finfo` call gets attributes about direct I/O support for the file.

Conclusion

Direct I/O requires substantially fewer CPU cycles than regular I/O on today's computers. I/O-intensive applications that do not get much benefit from the caching provided by regular I/O can improve performance by using direct I/O. The benefits of direct I/O will grow in the future as increases in CPU speeds continue to outpace increases in memory speeds.

***Bret R. Olszewski**, IBM Corporation, 11400 Burnet Road, Austin, TX 78758. Mr. Olszewski is a senior programmer working on MP performance. He joined IBM in 1989 and has worked on various aspects of AIX performance. He has a BS in Computer Science from the University of Minnesota.*

***David B. Whitworth**, IBM Corporation, 11400 Burnet Road, Austin, TX 78758. Mr. Whitworth is a staff programmer in AIX Performance. He has a BA in Computer Science from the University of Texas.*

AIX IP Security

By Kay Chang, Jackie Wilson, and Jason Wu

The Virtual Private Network (VPN) provides end-to-end secure connectivity over the Internet. AIX Version 4.3 has a VPN built-in to its operating system. AIX IP Security is a robust, IETF standards-conforming product implementing a VPN solution with easy-to-use, policy-based management.

The Internet has become a ubiquitous means of connectivity, and most companies now include the World Wide Web Uniform Resource Locator (URL) in their business addresses. Nearly everyone is now aware of the Internet, the enormous volume of information made available by both organizations and individuals, and the ability to retrieve that information.

Gradually, people have begun to view the loosely interconnected Internet as a means of extending the enterprise network. For example, remote business users can connect to their corporate network or to their business partners' network. The greatest challenge, however, has been to provide authenticity and privacy as needed, yet have the ability to use an insecure, open, public data network like the Internet.

The Virtual Private Network (VPN) is a mechanism to create a private "tunnel" over the public Internet that enables remote corporate users, branch offices, and business partners/suppliers to exchange information. The goal of a VPN is to provide end-to-end connectivity using the Internet, yet provide the dependability, security, and reliability that enterprises require in a private network. A VPN can also provide significant savings. A 1997 VPN Research Report by Infonetics Research, Inc. estimates that companies can save from 20% to 47% of their wide area network (WAN) costs by replacing leased lines with VPNs to remote sites.

AIX IP Security functions provide those VPN requirements since AIX® uses open, standard IP security functions including integrity, authenticity, and confidentiality. It interoperates with other IP security providers, and configures and manages its resources with a policy-based mechanism.

In addition to the IP security function, AIX IP Security also filters non-secure packets based on user-defined criteria (filter rule). This is primarily a firewall function where packet control between networks and machines is enforced.

What is AIX IP Security?

IP Security provides cryptography-based protection for all data at the IP layer for both IP Security Versions 4 and 6. It transparently provides secure communications, with no changes required to existing applications. Using robust cryptographic techniques, IP Security protects your data traffic in three ways:

- **Authentication:** Verifies the identity of a host or endpoint. Authentication algorithms provide proof of identity of the sender and data integrity. These algorithms use a cryptographic hash function to process a packet of data (with the immutable IP header fields included) with a secret key to produce a unique digest. On the receiver's side,

Kay Chang

Jackie Wilson

Jason Wu

the data is deciphered using the same function and key. If either the data has been altered or the sender's key is not valid, the datagram is discarded.

- **Integrity checking:** Ensures that no modifications were made to the data while in transit across the network. If any part of the data is modified during communication, the authentication check will fail.

- **Encryption:** "Hides" data and private IP addresses while in transit across the network in order to ensure privacy. Encryption uses a cryptographic algorithm to modify and randomize the data using a certain algorithm and key to produce "cyphertext." This makes the data unreadable while in transit. Once received, the data is recovered using the same algorithm and key using symmetric encryption algorithms. Encryption must occur with authentication to verify the data integrity of the encrypted data.

Tunnels

A *tunnel* is a virtual connection between two data endpoints. Tunnels can be used to set up a secure communication between two machines. Both sides must agree to the security parameters before the tunnel will work. This is known as setting up a security association.

The security association can use either static, manually distributed keys, or automatically refreshed keys. To comply with Internet Engineering Task Force (IETF) standards, all implementations must use manual keying. This ensures a base set of cryptographic capabilities that will allow IP security implementations from different vendors to interoperate.

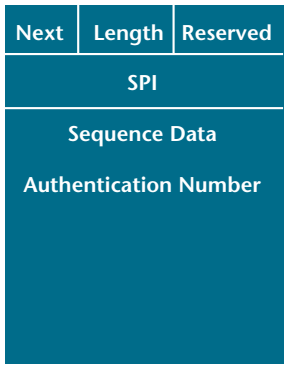
IBM has developed a method known as *IBM tunnels* for automatically refreshing keys. Packets sent and received on an agreed-upon port number negotiate the security association parameters and automatically refresh keys.

The IETF is currently developing a standard for implementing automatic key management known as ISAKMP/Oakley. This protocol, once fully defined and implemented, will replace IBM tunnels. IBM tunnels are currently implemented in AIX Firewall beginning with Version 2.1 and in AIX 4.3. It can be used today for automatically refreshing keys.

Evolving Standards

IP Security protects data by using the Authentication Header (AH) or Encapsulating Security Payload (ESP) header and encrypted payload inserted in the IP packet. The authentication header can be used by itself or in combination with the ESP header. In the first RFC versions of the IP security headers, authentication data was sent using an AH. Encrypted data, or cyphertext, was sent in a separate ESP header. In 1997, the ESP header format was extended to include a field for the authentication digest to enable both encryption and authentication information to be processed in one header. In addition, both AH and ESP formats were extended for replay protection by including a sequence number that prevents old replay packets from being accepted. Figure 1 shows the AH format and Figure 2 shows the ESP format for the new IETF headers.

Authentication Header



AH Format Features

- **NEXT** is the 8-bit "next protocol" field from IP
- **LENGTH** is the length of this header in 32-bit words
- **SPI** is a 32-bit field identifying this security association
- **SEQUENCE NUMBER** is a 32-bit counter used to detect and reject replayed packets
- **RESERVED** is a 16-bit field, currently all zeros
- **AUTHENTICATION DATA** contains the value computed by the authentication algorithm, padded to an integral multiple of 32 bits

Figure 1. Authentication header for new IEFT headers

A new Hashed Message Authentication Code (HMAC) standard for strengthening the security of authentication algorithms was adopted in RFC 2104. IP security support for AIX 4.3 implements the RFC versions of the headers for backward compatibility and the new draft standards. Supported authentication algorithms in AIX 4.3 are HMAC MD5 (Message Digest 5, RFC 1321), HMAC SHA1 (Secure Hash Algorithm) and Keyed MD5 (for backward compatibility). The authentication algorithm determines the header format.

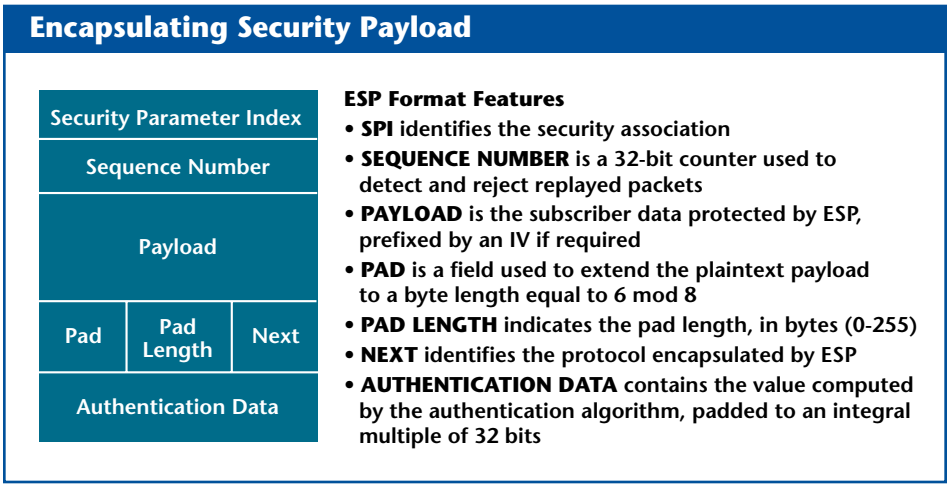


Figure 2. ESP format for new IETF headers

Encryption algorithms include DES_CBC_8 (Data Encryption Standard Cypher Block Chaining), DES_CBC_4, and an exportable version of DES known as Common Data Masking Facility (CDMF). AIX 4.3 IP Security also supports the ability to send authentication data in the ESP header, sometimes referred to as DES_CBC_MD5.

Filtering Capability

Filtering is a function in which incoming and outgoing packets can be accepted or denied based on a variety of characteristics. This allows a user or systems administrator to configure the host to control the traffic between this host and other hosts. Filtering is done on a variety of packet properties, such as source and destination addresses, IP Security Version (4 or 6), subnet masks, protocol, port, routing characteristics, fragmentation, interface, and tunnel definition.

Rules known as filter rules associate certain kinds of traffic with a particular tunnel. In our basic configuration, when a user defines a tunnel, filter rules are automatically generated to direct all traffic from that host through the secure tunnel. If more specific types of traffic are desired, the filter rules can be modified to allow precise control of the traffic using a particular tunnel.

Similarly, when the tunnel is modified or deleted, the filter rules for that tunnel are automatically modified or deleted. This greatly simplifies IP Security configuration and helps reduce human error. Tunnel definitions can be propagated and shared among AIX machines and AIX Firewall using import and export utilities.

Filter rules are necessary to associate a particular type of traffic with a tunnel, but data being filtered does not necessarily need to travel in a tunnel. This allows AIX to provide base firewall function for users who want to restrict the flow of certain types of traffic to or from their machine. This is especially useful for the administration of machines in an intranet or for those that do not have firewall protection.

Dual Stack Support

AIX IP Security Versions 4 and 6 implement two separate stacks. This enables the IP Security function to be configured independently. The IP Security function can be enabled, disabled, or modified on one IP Version without affecting the behavior of the other. (See Figure 3).

Configuring AIX IP Security

Since all IP traffic will get filtered, it is important to set the default filtering behavior before IP Security is started. The default rule will determine whether traffic that does not match any prior filter rule will be permitted or denied. In a client scenario where a user wants to set up a secure tunnel, but otherwise not use filters, the default action

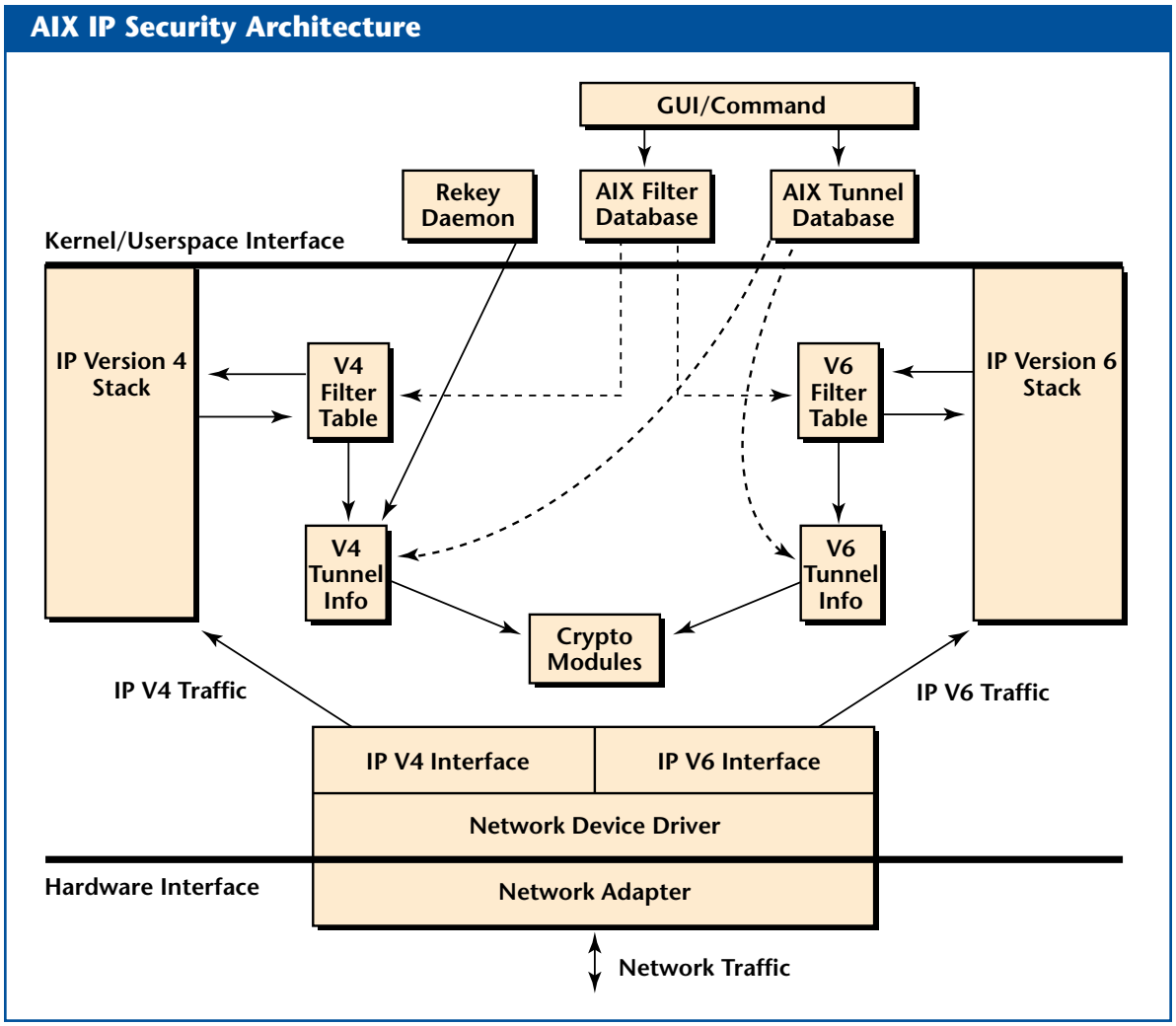


Figure 3. Architecture for AIX IP Security

would be permit. In the case where filter rules were being used to protect incoming and outgoing traffic, such as in a firewall or intermediate gateway, the default action would be deny. The default action is selected using the SMIT Start/Stop IP Security panel shown in Figure 4.

Once the default behavior is set, the user can define tunnels. To define a tunnel, the user must know the IP address of the remote host, whether the remote host is behind a firewall, and how the data in the tunnel should be protected. The tunnel may provide authentication, or both authentication and encryption. If the security characteristics of the remote side are known, then the user simply matches the characteristics of the remote and inserts the appropriate IP address information. If the user is setting up the initial end of the tunnel, then the user can select the characteristics desired.

For an IBM tunnel, the user specifies the IP addresses of the endpoints of the tunnel and firewall (if applicable), the encryption algorithm (if encryption is to be used), the life time of the session key, the key refresh time, and whether that

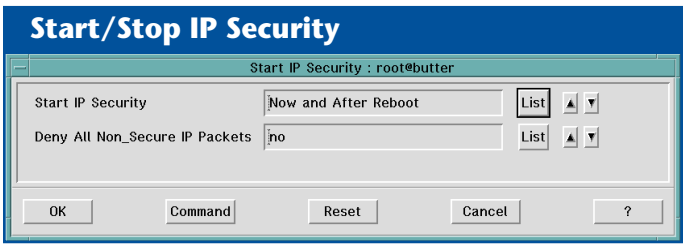


Figure 4. Start/Stop IP Security

host will act as the initiator of the key exchange communication.

For a manual tunnel, the user specifies the IP addresses of the endpoints and firewall (if applicable), selects the AH and ESP algorithms, and the destination Security Parameter Index (SPI) value. Together with the IP address, this uniquely identifies the tunnel. The keys can be manually entered or automatically generated.

Due to export regulations, U.S. and Canadian customers get a higher degree of encryption (for example, DSS), whereas other countries have their own import restrictions.

SMIT Configuration Path

Figures 5 and 6 show examples of setting up two types of manual tunnels. One tunnel is connected to a remote system 9.3.97.112 using SPI value 500. The other tunnel is connected to a remote system 9.53.159.251 behind a firewall using SPI value 501. Both use an AH/ESP combination with HMAC MD5 for authentication and DES_CBC_8 for encryption. The data will be encapsulated in the draft standard versions of the AH and ESP headers. Since this is a manual tunnel, the remote machine can be any vendor's machine running IP Security with the draft standard version of AH and ESP. The keys are manually exchanged.

Both tunnels have automatically generated filter rules, and all traffic between the tunnel endpoints will be sent through the tunnel. Any packets that are not protected by the specified security parameters are discarded, and all packets will be authenticated and encrypted.

Additional filter rules can be specified by using any of the modifiers such as IP Source and destination masks, protocol, port number, routing, interface, and fragmentation control. Operators, such as less than, greater than, equal to, and not equal can be used on the port number fields. This provides great flexibility and control in specifying data to be protected in a tunnel or to be permitted or denied from flowing outbound or inbound. See Figure 7 for an example of the filter rules that were automatically generated.

Once the filter rules are generated, they are stored in a table and loaded into the kernel. When packets are ready to be sent or received from the network, the filter rules are checked in the list from top to bottom to determine whether the packet should be

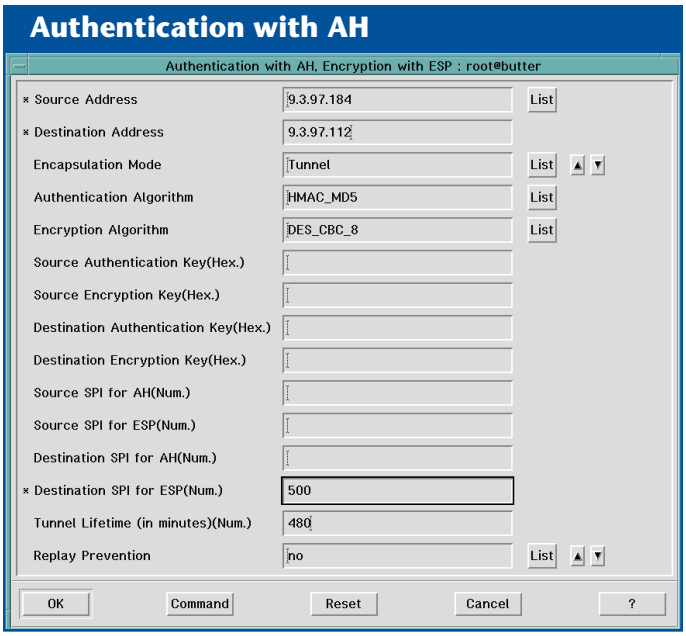


Figure 5. Using AH for authentication

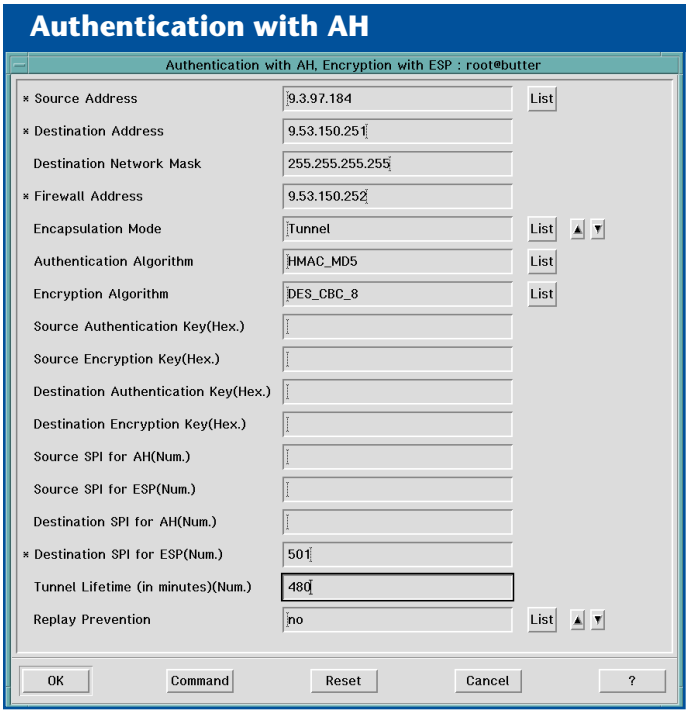


Figure 6. AH authentication

permitted, denied, or sent through a tunnel. The rule criteria are compared to the packet characteristics until a match is found or the default rule is reached. Figure 8 shows a listing of the filter rule.

The first three general rules (not specific to any configured tunnel) allow AH, ESP, and IBM tunnel session traffic to flow. Rules 4 and 5 are the rules for tunnel number 1, which is the first manual tunnel shown earlier (see Figure 5). Rules 6 and 7 allow traffic for tunnel number 2, as shown in Figure 6. Rule number 0 (last rule to match), the default rule, shows the permit default action on packets that do not match any prior rules.

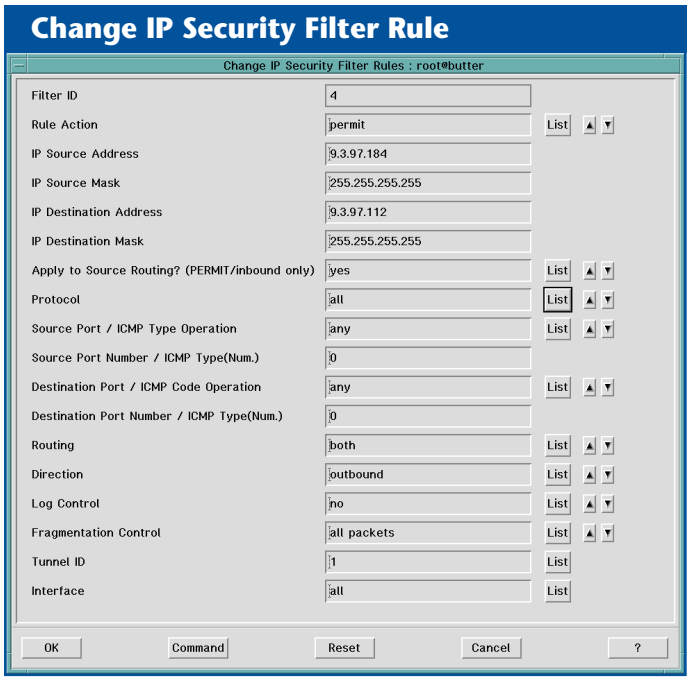


Figure 7. Filter rule for change IP security

VPN Scenarios and IP Security Solutions

Three obvious VPN scenarios today include:

- Remote and secure access
- Extranet with business partner or supplier intranet
- Branch office intranet

Figure 9 highlights these scenarios.

Remote Secure Access

The remote access environment is one in which a mobile computer is connected to the corporate network. These connections are primarily handled by dial-up links, such as point-to-point protocol (PPP). Multiple

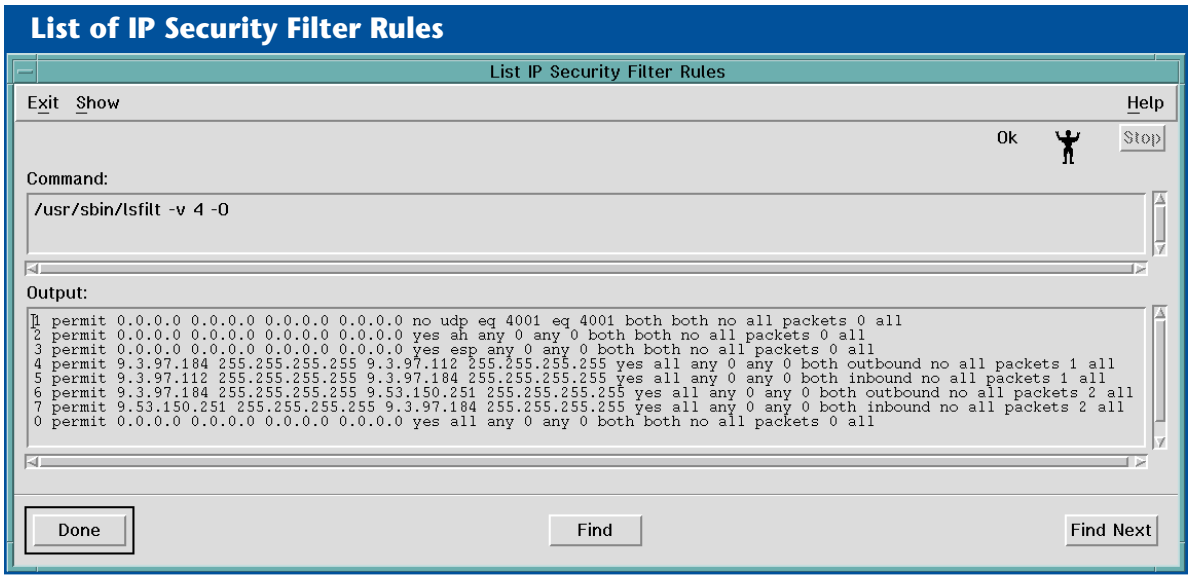


Figure 8. Filter rules for IP security

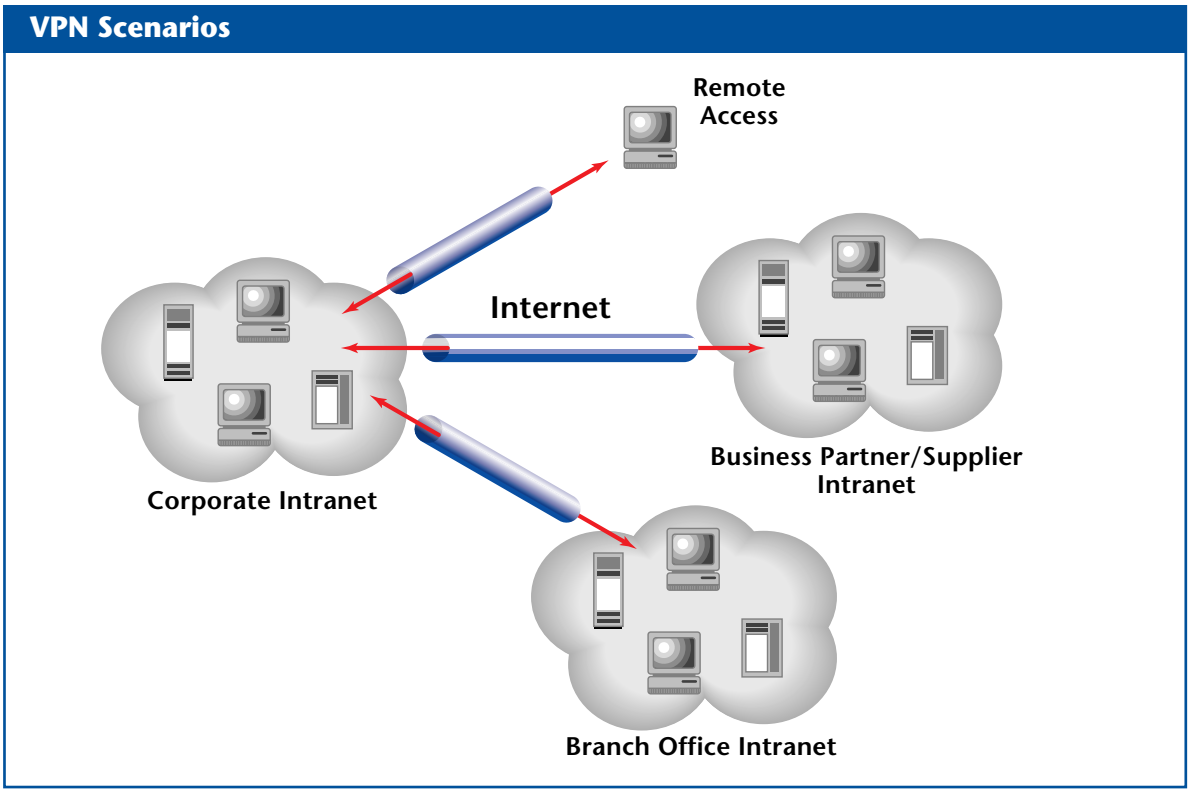


Figure 9. VPN Scenarios

modem banks provide the incoming connection, often with long distance carrier charges.

The IP security function of VPN lets users dial the local calling number of the Internet Service Provider (ISP), where end-to-end security is achieved by creating a separate tunnel between a remote mobile computer to the server inside the corporate network. This is end-to-end tunnel flow through the ISP and corporate firewall to the destination server.

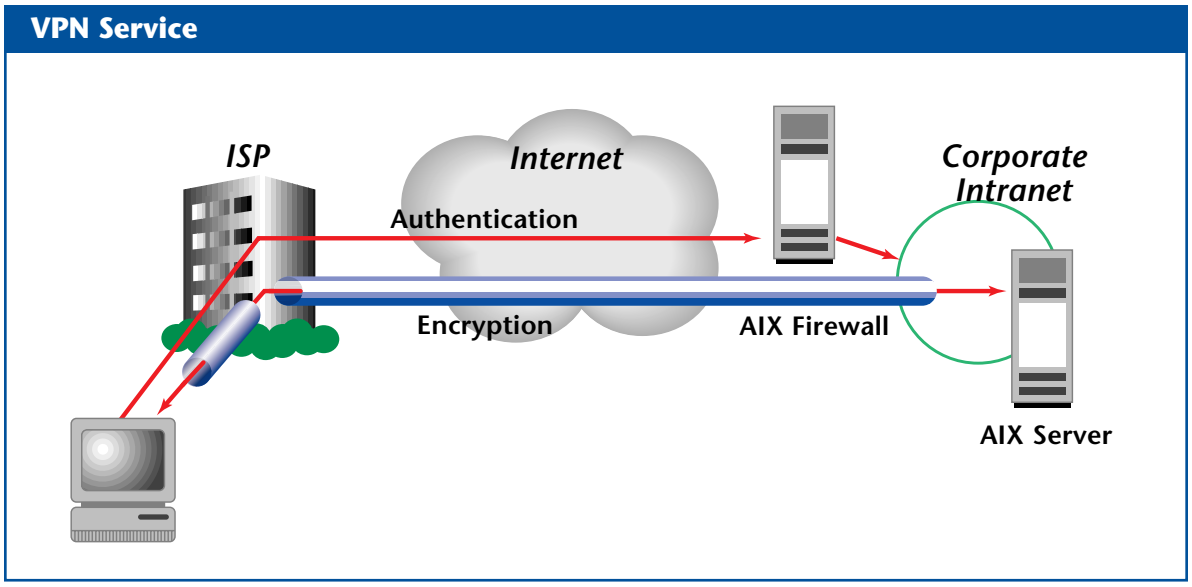


Figure 10. VPN Service

Extranet with Business Partner/Supplier

The extranet environment interconnects the corporate intranet to intranets from business partners and/or suppliers via the Internet. Many corporations have created and maintained a separate security policy and a private data network. Often when companies merge or when they must communicate with suppliers, the setup and operational costs are prohibitably high for smaller business partners because of the high costs of maintaining the network charges (such as leased T1 lines). The VPN based on IP Security provides a cost-effective, secure, end-to-end communication over the Internet.

Branch Office Intranet

A branch office intranet is a secure interconnection between the branch intranet and the corporate intranet. Unlike the supplier network, it is expected to have a similar setup in types of networks and applications. It generally requires expensive leased-line connections or on-demand dial connections to the home office. A VPN with IP Security provides 24-hour ease-of-use connectivity via inexpensive Internet links.

Solutions with IP Security

Each scenario described above highlights the problems facing businesses today. AIX IP Security addresses these situations with authenticity and confidentiality. It works with most firewall products to support popular mobile or desktop clients with an IP security solution.

Conclusion

The VPN provides end-to-end secure connectivity over the Internet. AIX IP Security is a robust, IETF standards-conforming product implementing a VPN solution with ease-of-use, policy-based management.

VPN is built-in to both AIX Version 4.3 and the IBM Firewall product. Popular client products, such as Microsoft® Windows™, have available offerings from IBM add-on or third-party vendors.

One important consideration is that using the Internet as a dedicated private network requires a service level agreement (SLA) to guarantee quality of service that the Internet lacks. Vendors must develop a service to provide functions that the business demands and expects. Since the Internet consists of loosely interconnected heterogeneous networks, it only guarantees a service level at best basis. Service providers, such as IBM Global Services, must add extra services for the VPN to provide availability, performance, and reachability. IBM began the SLA initiative to help solve these problems faced by companies using the Internet.

***Kay Chang**, IBM Corporation, 11400 Burnet Road, Austin, TX 78758. Ms. Chang is a senior programmer in AIX communication architecture working on AIX Connections and the Network Computer. She has an MS in Computer Science from Wright State University in Dayton, Ohio.*

***Jackie Wilson**, IBM Corporation, 11400 Burnet Road, Austin, TX 78758. Ms. Wilson is the project lead and a programmer on the AIX implementation of IP Security shipped in AIX 4.3. Since joining IBM in 1982, she has programmed and led projects in wide area networks, developed device drivers and adapter software for X.25, SDLC, bisync and modem support. She has a BSEE in Electrical Engineering and Computer Science from Princeton University and an MBA from St. Edwards University. She is also the inventor of three software patents in network programming.*

***Xinhua (Jason) Wu**, IBM Corporation, 11400 Burnet Road, Austin, TX 78758. Mr. Wu is an advisory programmer with AIX Development. He has an MS in Computer Science from the University of Southern California.*

Novell Network Services 4.1 for AIX

By Denise Genty and Rakesh Sharma

Novell Network Services 4.1 for AIX is a NetWare Server based on Novell's Cross-Platform Services 4.10b. Novell Network Services (NNS) is enterprise network operating software that enables a workstation to share files, printers, applications, and other resources among client PCs running operating systems like OS/2® and Windows. NNS is a Licensed Program Product (LPP) that operates on AIX Version 4.2.1 or 4.3.

Novell® Network Services includes Novell Directory Services (NDS), which provides a global naming services that is distributed across the entire NetWare® network. NNS places NetWare Services on an AIX® computer, turning AIX into a non-dedicated NetWare server that provides both host operating and application services and NetWare Services. NetWare Services runs transparently on AIX, so clients are not aware whether the server is a native Novell NetWare server or an AIX NetWare server.

NNS integrates the AIX computer world with the networked PC world. It also solves the need for access to company-wide applications, such as databases or schedules, that run on host systems. Users can write reports and track data using PC desktop applications while having access to the AIX system resources.

NNS Features

NNS provides file and print system services to PC clients. The security model that it supports prevents unauthorized access to network resources. For network management, NNS includes statistical utilities, Simple Network Management Protocol (SNMP), and AIX System Management Interface Tool (SMIT) menus. With NDS, users can log in to the network rather than to an individual server. For application developers who develop applications for NetWare, NNS includes the NetWare application programming interface (API) and APIs for the NetWare IPX, SPX, and SAP protocols.

It includes new features of a Lightweight Directory Access Protocol (LDAP) server, IP tunneling, and licensing models, which are described below.

Novell Directory Services

Novell Directory Services has a global naming service that is distributed across the entire NetWare network with a single point of server administration. NetWare users can login to the Directory tree and, with appropriate rights, access any resource on the network, regardless of physical location.

NDS provides the following features:

- Serves as a distributed information service that stores many types of data

Denise Genty

Rakesh Sharma

- Provides scalability and reliability through its ability to be partitioned and replicated
- Provides a single point of identification to the network with its single sign-on authentication methods
- Uses access control lists (ACLs) to access services
- Enables application developers to customize the information contained in the Directory with its extensible schema
- Uses X.500 as its model

NDS replaces the NetWare bindery found in NetWare Versions 2.0 and 3.0. It allows users to login to the network rather than to an individual server. Its flexibility enables it to work well in both small workgroups with a few servers and corporate organizations with hundreds of servers.

NDS also includes time synchronization. This means that all servers within the same tree report the same time and make automatic adjustments, even when they are located in different time zones.

Other services include the following:

File System Services: Supports multiple name spaces and a multilevel file access system. The namespace feature allows users to view filenames in the naming conventions of their workstation's operating system. Trustee assignments control file access to users and groups, inherited rights, and file attributes, which can restrict rights to specific files.

Print Services: Allows NetWare clients to access printing resources on both NetWare and AIX printers. AIX can be used as a print server and NNS can be used as a print queue server to Novell NetWare network printers. NNS supports all NetWare printing protocols.

Security Services: Provides secure NDS authentication with private-key/public-key encryption login restrictions. The keys are strings of numbers used in mathematical functions. The client workstation uses a private key to encode messages sent to the NetWare server. The server uses a public key to decode the messages. Neither of the two keys nor the user's password are sent across the network.

Online Publications

Fifteen publications are shipped with NNS. One book, *Quick Beginnings*, is shipped with NNS as a hardcopy publication. After installing the `ncps.html.en_US` image, the books are available in PDF format in the `/usr/share/man/info/en_US/a_doc_lib/nns` directory. They can be viewed online using Adobe Acrobat® Reader. You can use a Web browser at the following URL to view a directory page from which you can access the PDF documentation: *file:///usr/share/man/info/en_US/a_doc_lib/nns/nnsnav/topnam.htm*.

Figure 1 shows the NNS documentation home page.

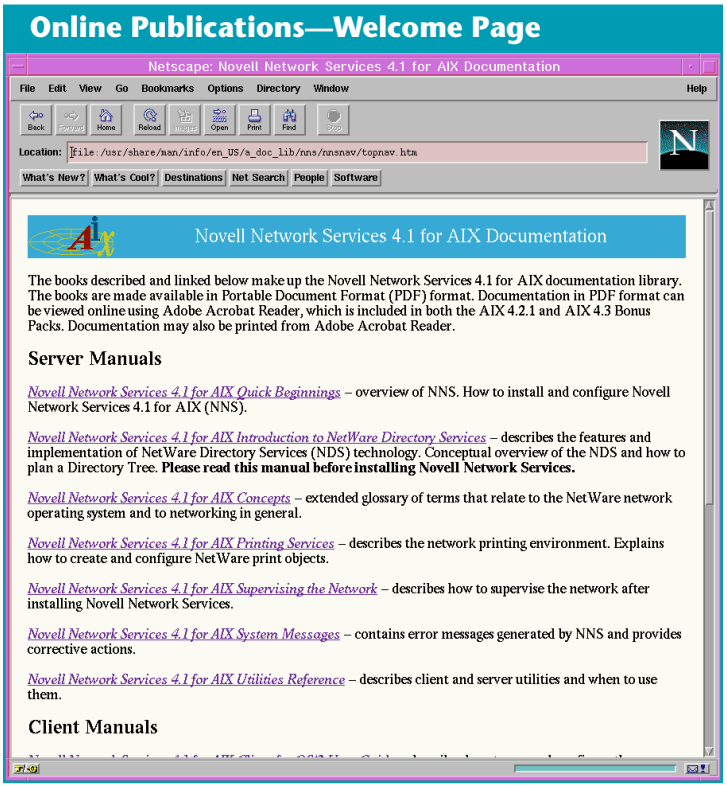


Figure 1. Welcome page for Online Publications

Communications Protocols

NNS supports IPX and IP tunneling transports. NNS supports IPX packet-bursts (as opposed to NCP request responses for every request). This improves file-server data transfer performance.

NNS IPX/SPX includes support for SPX II in addition to SPX protocol support. SPX II delivers much higher performance than SPX I for connection-oriented transport. Since protocol dialect is negotiated at connection time, NNS IPX/SPX will interoperate only with the clients supporting SPX I.

NNS supports IP tunneling for IPX, a feature that provides IP protocol capability to NNS 4.1. IPX packets are sent over an IP network encapsulated in a User Datagram Protocol (UDP). This feature requires both Domain Name Server (DNS), which is part of AIX, and Domain SAP/RIP Service (DSS). The tunneled IPX network requires DSS.

NetWare IP requires its own domain, but AIX supports IP Security to complement NetWare IP. IP Security allows secure IPX tunneling across the Internet.

API Support

The NetWare API for C is the interface to the C programming libraries for workstation applications. These libraries enable applications to interact with the NetWare server and access NDS.

LDAP Server

Lightweight Directory Access Protocol (LDAP) is a directory service protocol that runs over TCP/IP. It is used primarily to search for information in the directory, where the directory service protects the data and provides the security for the data.

NNS LDAP support is compatible with LDAP Version 2. NNS 4.1 LDAP is compatible with AIX LDAP Client support, which includes the Java™ Class Library and an AIX shared library. In addition to the AIX client, NDS/LDAP is compatible with any client that supports LDAP Version 2 protocol.

An example of the use of the NNS LDAP server would be to obtain NDS users' information, such as e-mail addresses, using an LDAP-enabled Web browser.

Licensing Models

NNS uses a client/server licensing model. Specifically, it uses the License User Management System, based on the iFOR/LS management system. This iFOR/LS server-based licensing scheme consists of a license server that maintains a license database. This database contains the availability of NNS licenses and maintains a usage count of the product.

> ***NDS flexibility enables it to work well in both small workgroups with a few servers and corporate organizations with hundreds of servers.***

Two types of licenses can be purchased for NNS: SCALE server license and user licenses. If you do not purchase any licenses, you may configure a SANDS mode server, which has a default of two user licenses.

A SANDS tree can be installed and configured on a single server. No other servers can be added to the Directory, and the NDS database cannot be partitioned or distributed. A SCALE server license is used when installing and configuring the NDS tree. A SCALE tree allows server database replication to multiple servers, and the NDS database can be partitioned and distributed. The directory tree can be shared with other servers.

Minimum Server Configuration

After the license server is configured, it takes four steps to configure the AIX NetWare server. The first step is to configure the maximum number of user licenses. Using the SMIT tool, enter the maximum number of licensed connections, as shown in Figure 2. Next, configure the server name and unique IPX internal address on the Minimum Configuration screen, shown in Figure 3. Verify that the IPX LAN interface data is correct on the Change/Show a LAN Interface screen as shown in Figure 4.

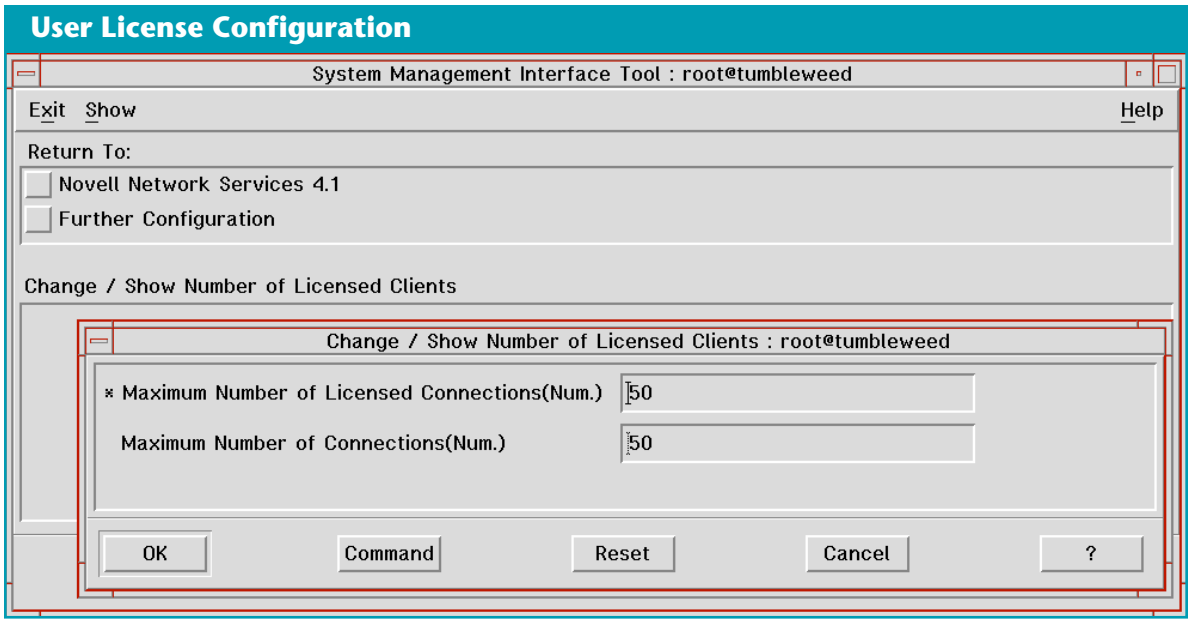


Figure 2. Configuration of user licenses

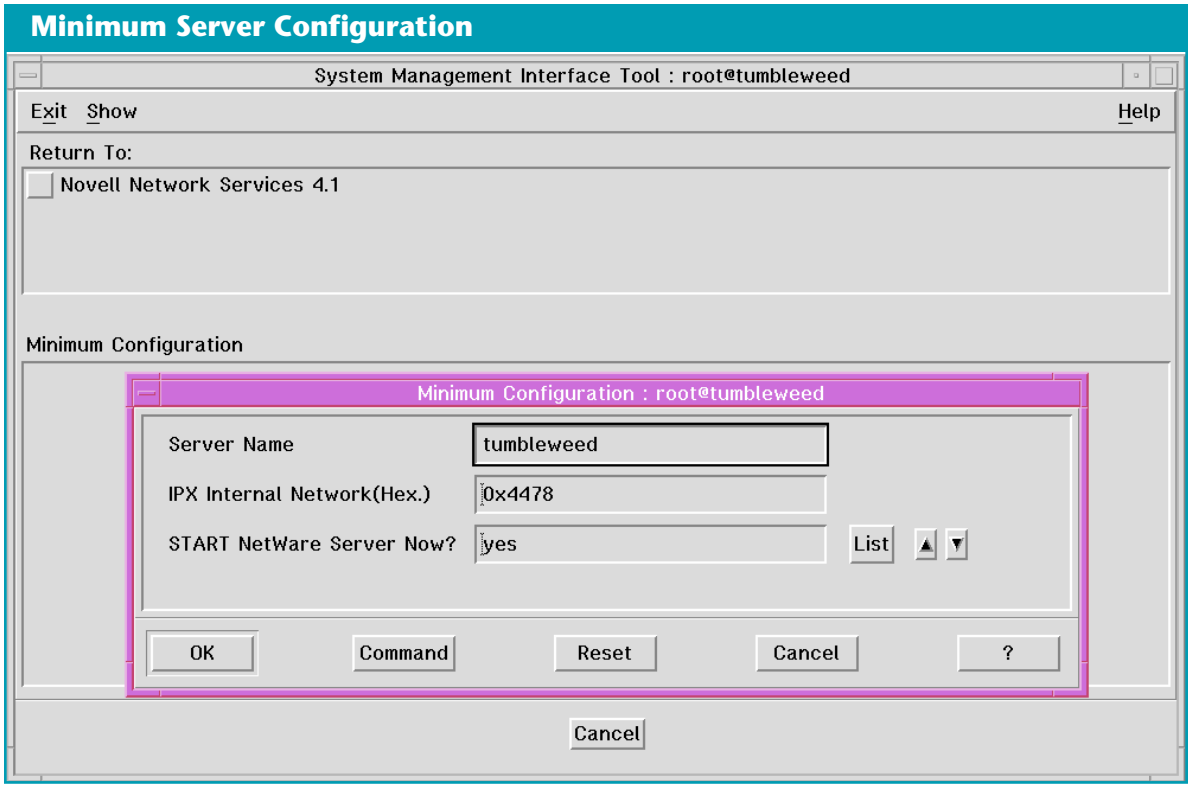


Figure 3. Minimum server configuration

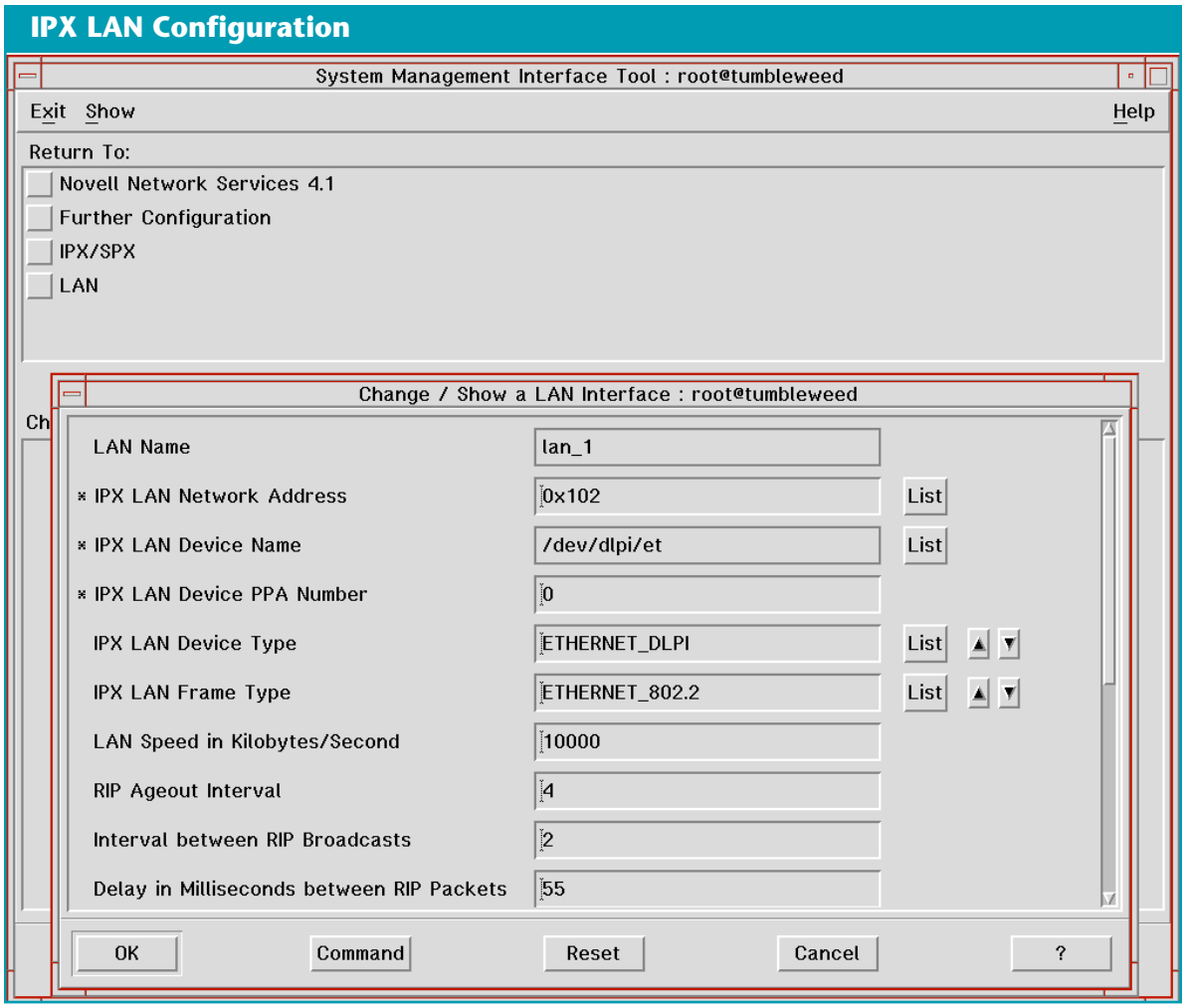


Figure 4. IPX LAN configuration

Finally, install the NDS tree and configure the tree name, context, and administration user password using the `SMIT` option; then configure Network Directory Services using `dsinstall`. The AIX NetWare server is now ready for operation with one volume (`SYS:`) and one user (`admin`) created.

The Structure of NNS

Figure 5 shows the overall architecture of NNS. Modules within the dark box are available only when the NetWare server is up. Modules outside the box are either required by the NetWare protocol stack or run on the stack independent of NNS. The ovals represent daemons or AIX processes. The rectangles represent kernel drivers. The lines, which represent the significant paths of communication, are not meant to be inclusive. When daemons overlap (as the SAP and Diagnostic daemons in Figure 5), the daemon in the back has direct lines of communication to the same drivers as the daemon in front. For example, the Diagnostic daemon has lines of communication to the IPX, RIPX, and SPX II drivers.

NNS uses two key daemons to initialize NetWare Services:

- NPS daemon is primarily responsible for linking kernel drivers and spawning daemons required to set up IPX communication services.
- NetWare daemon is responsible for linking kernel drivers and spawning daemons required to set up NNS services.

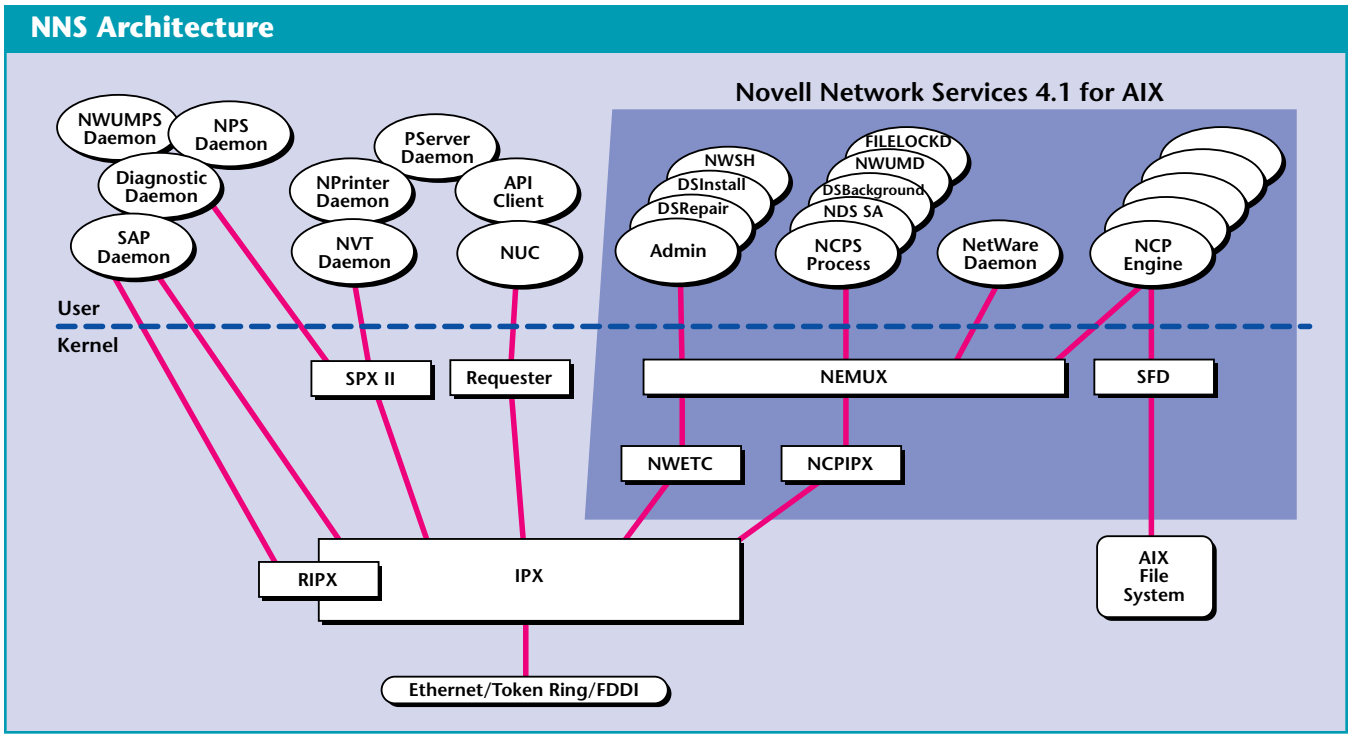


Figure 5. NNS architecture

NDS Example Configuration

A NetWare server can be configured to have multiple volumes in addition to the SYS volume; NDS can be configured so that a business' network has one or multiple directory trees. Companies with centralized management usually prefer a single tree. Businesses with decentralized management may prefer multiple trees if there is little interaction between groups, or a single tree if there is much interaction between groups. Figure 6 shows an example of a directory tree and its objects.

In the tree example in Figure 6, the NDS tree contains a top-most organization named

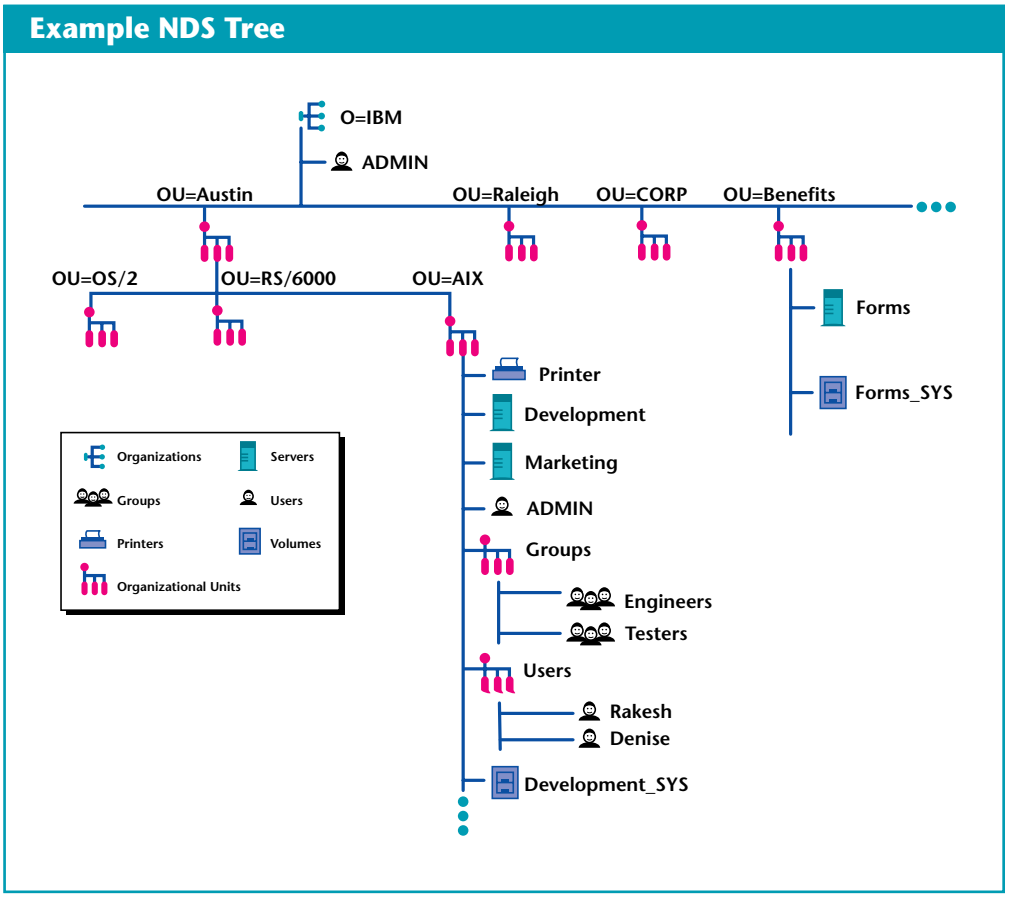


Figure 6. Directory tree and its objects

"IBM." There is one user, ADMIN, which has been given all rights to the tree. Those rights flow down to the organizations of Austin, Raleigh, Corporate, and Benefits. Company-wide resources can be accessed and administered, even if the resource is in a different geographical location. End users have access to their organizational unit or other units without requiring logins to individual servers.

This tree is divided by geographic locations and divisional functions. There is a logical organization for the Austin and Raleigh IBM sites, and organizations for corporate headquarters and the benefits organization. To restrict access, the Austin organizational unit has three different divisions: OS/2, RS/6000, and AIX. In this example, a second ADMIN user is defined in the AIX organization. ADMIN, the administrator for the organization, has all rights to the volumes and servers in AIX. This ADMIN user can administer the day-to-day operations of the AIX organization.

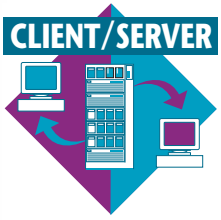
The AIX organization has defined one NetWare printer, two servers, and a volume named `Development_SYS`. The Development and Marketing organizations each have one server primarily for their use. There are groups defined for engineers and testers. The Users group contains users named Rakesh and Denise, who have access to the AIX printer and the two servers.

Groups are a beneficial NDS feature because the administrator can assign users to a group, then, with just one trustee assignment, grant access to all the users who belong to the group. In this example, the Benefits organizational unit has a server that contains all benefit claim forms for the IBM company. On the `Forms_SYS` volume, a user such as Denise in the AIX organization could be given access rights to the volume to print necessary claim forms.

***Denise M. Genty**, IBM Corporation, 11400 Burnet Road, Austin, TX 78758. Ms. Genty is a staff software engineer in the RS/6000 Division. She has worked in IBM RS/6000 hardware and AIX software development for seven years. She has concentrated on AIX device system configuration and communications applications, and currently works on NetWare for AIX. Ms. Genty has a BS in Computer Science from Texas A&M University.*

***Rakesh Sharma**, IBM Corporation, 11400 Burnet Road, Austin, TX 78758. Mr. Sharma is senior software engineer with a specialty in networking protocols, TCP/IP, and UNIX® interoperability with PCs. He is currently working with Windows NT/PC interoperability for AIX. He has a BS in Electrical Engineering from ITT Kanpur in India and an MS in Computer Science from North Dakota State University.*

Developing a Java Client/Server Prototype



By Dr. Peter W. Farrett

This article outlines general steps for developing a Java-based client/server application. The client and server discussions highlight important concepts and techniques that are fundamental for developing Java™ client/server applications. The article also includes complete source code for the application described.

Client/server applications generally consist of processing queries where the client utilizes services provided by the server. Communication must be established and no loss of data can occur.

In the client/server application described in this article, the client maintains the user information (member data) and the server handles the requests for information. Specifically, the application will read from the client and write data to a user member file via the server with appropriate server system messages. The server will also send user information back to the client.

Every Java client/server application, including the one described in this article, contains several important components:

- **Graphical user interface (GUI):** A Java component that is implemented with a subclass of the Abstract Windowing Toolkit (AWT) component class. In our example, the client has a simple GUI consisting of a text field, string, and several buttons and labels; the server side does not require a GUI.
- **Input and output streams:** Java file I/O. These streams must read both the client and server data. For this application, we used the Java package, `java.io`, which contains input and output streams (read/write). The `InputStream` and `OutputStream` classes (in `java.io`) are abstract superclasses that define I/O behavior.
- **Sockets:** Both the client and server programs require network communication. These low-level methods "bind" each other via a communication link—from endpoint to endpoint. The `java.net` package implements the client- and server-side with `Socket` and `ServerSocket` classes, respectively.
- **Multithreading:** A thread is a single control flow (execution) within a program. In our application, we implement the `start()`, `stop()`, and `destroy()` methods relative to our threads of execution since numerous, simultaneous threads will occur based on client requests.

The Client (Applet)

The client must read and write input data. The `loadMember()` and `writeMember()`

Peter W. Farrett

methods handle this requirement along with socket connection objects. The client must first obtain the server's connection. For example, `writeMember()` method is accomplished by the `socket` object, a string that identifies the hostname, and port address, which creates a new socket. Figure 1 shows how the client creates a new socket.

```
socket s = null;
theServer = "austin.ibmtest.com";
thePort = 8081;

s = new Socket(theServer,thePort);
```

Figure 1. Creating a new socket

In order to read member data from the server's file system, the client must access Java I/O streams that relate to the socket connection. This "connects" `socket s` with the Java `input` stream. The subsequent lines of code wait for the server to read a file, then send back user information, as shown in Figure 2.

Conversely, Figure 3 shows how to write data from the client (to be written at the server's location via `ServerSocket`) and send the request to the server for processing.

Once the socket is connected with the `getOutputStream()` method, user information (`companyname`, `addr1`) is sent out through this pipeline. The `flush()` method follows each `println()`, because flushing is necessary after each statement. Finally, file termination (end of transmission) denotes an end of file (EOF) delimiter.

The remaining code implements other Java features, such as error handling, the proper closing of I/O streams, and event-driven actions. The main function of this client example is to request user information from the server, send user information to the server, and terminate the transaction at completion. See complete source code at the end of this article.

The Server (Application)

One fundamental task of a server is to accommodate requests from multiple clients without halting, blocking, or stopping a client request. This involves two important concepts:

- The Java `ServerSocket` object (in `java.net`) is used so the server listens to a specific port.

```
DataInputStream in = new DataInputStream(s.getInputStream());

line = in.readLine().trim();
while(!line.equals("end of transmission"));
{
  text.appendText(line + "\n");
  System.out.println(line);
  line=in.readLine().trim();
}
```

Figure 2. Reading data from the server

```
PrintStream out = new PrintStream(s.getOutputStream());
out.println(companyname);
out.flush();
out.println(addr1);
out.flush();
out.println("end of transmission");
out.flush();
```

Figure 3. Writing data to the server

```
ServerSocket s Socket;
...
...
while(running) {
  Socket soc = null;
  try {
    soc = sSocket.accept();
    serverRequest(soc);
  }
  catch(IOException error){}
}
```

Figure 4. Requests from multiple clients

- The server multithreads all client requests, as shown in Figure 4. A simple loop handles client-side requests.

The `serverRequest()` call creates a new server object and immediately starts a single thread for each client request and invokes the Java `run()` method. Similar (client) coding techniques are used with respect to our previous use of I/O streams, `println()` and `flush()` methods; however, file I/O is added for reading and writing member information to a file, as shown in Figure 5.

It is important to remember to close all files and their associated streams in the order in which they occurred after I/O and file processing. Figure 6 shows an example. Also, error handling should be implemented when appropriate.

Finally, Figure 7 shows how we could override the Java `stop()` method with our own `stopThread()`.

```
in.close();   //file #1 for read
out.close();  //stream associated with file #1
out2.close(); //file #2 for read/write
in2.close();  //stream associated with file #2
```

Figure 6. Closing files and associated streams

```
public void stopThread(Thread ssThread)
{
  if (ssThread != null)
  {
  ssThread.stop();
  ssThread.destroy();
  }
}
```

Figure 7. Overriding Java's stop() method

Invoking a `stop()` and `destroy()` method accomplishes two things:

- It ensures that the running thread has stopped, based on a client transaction.
- It frees system resources. See the complete source code at the end of this article.

Complete Source Code

The following sections provide complete source code for the Java client/server application.

Figure 8 shows the client code, Figure 9 shows the server code, and Figure 10 shows the HTML code.

```
File file = new File("memberinfo.txt");
RandomAccessFile in = new
                      RandomAccessFile(s.getOutputStream(file,"r"));
PrintStream out = new PrintStream(s.getOutputStream());

DataInputStream in2 = new DataInputStream(new BufferedInputStream
                          (s.getInputStream()));
File file2 = new File("member" + counter + ".txt");
RandomAccessFile out2 = new RandomAccessFile(file2,"rw");
```

Figure 5. Reading and writing member I/O

```
import java.awt.*;
import java.applet.*;
import java.io.*;
import java.net.*;


public class ProtoClient extends Applet
{
  String theServer;
  int thePort;
  Label title = new Label("SDP Registration Form", Label.CENTER);
  Label inputField1 = new Label("Name");
  Label inputField2 = new Label("Address");
  TextArea text = new TextArea();
  Button submit = new Button("Submit New Member Info");
  String companyName, addr1;

  TextField companyname = new TextField("",10);
  TextField addr1 = new TextField("",10);

  public void init()
  {

        add(title);
        add(text);
        add(inputField1);
        add(companyname);
        add(inputField2);
        add(addr1);
        add(submit);

addNotify();
resize(411,322);
  }


void loadMember()
{
Socket s = null;
theServer = "austin.ibmtest.com";
thePort = 8081;

try {
    s = new Socket(theServer, thePort);
System.out.println ("client found, waiting for a connection...") ;
java.io.InputStreamReader(s.getInputStream())) ;
DataInputStream in  = new DataInputStream(s.getInputStream());
String line = in.readLine() ;
text.appendText(line + "\n") ;
System.out.println("connected") ;

int counter = 0 ;

//Wait for server to read a file and send each line
line = in.readLine().trim() ;
          while(!line.equals("end of transmission"))
          {
```

(continued on following page)

Figure 8. Client code

(continued from previous page)

```
            text.appendText(line + "\n");
            System.out.println(line) ;
            System.out.println(counter++) ;
            line = in.readLine().trim() ;
           }

System.out.println("sleeping for 2 seconds...") ;
try
{
Thread.sleep(2000) ;
}
catch(InterruptedException e2)
{
    System.out.println(e2.toString()) ;
}

System.out.println("waking up...") ;
java.io.OutputStreamWriter(s.getOutputStream())) ;
PrintStream out = new PrintStream(s.getOutputStream());
out.println("end of transmission") ;
out.flush() ;
System.out.println("finished writing data") ;

     }
     catch (IOException e) {
        showStatus("Server not activated: " + e);
     }
     finally {
        if (s!=null) try { s.close(); } catch (IOException e) {}
     }
  }


  void writeMember()
  {
     Socket s = null;
     theServer = "austin.ibmtest.com";
     thePort = 8081;

     CompanyName = companyname.getText();
     Addr1 = addr1.getText();

     try {
       s = new Socket(theServer, thePort);
System.out.println ("client found, waiting for a connection...") ;
java.io.InputStreamReader(s.getInputStream())) ;
DataInputStream in  = new DataInputStream(s.getInputStream());
String line = in.readLine() ;
text.appendText(line + "\n") ;
System.out.println("connected") ;

int counter = 0 ;

//Wait for server to read a file and send each line
line = in.readLine().trim() ;
```

(continued on following page)

Figure 8. Client code

(continued from previous page)

```
        while(!line.equals("end of transmission"))
{

       text.appendText(line + "\n");
System.out.println(line) ;
line = in.readLine().trim() ;
}

System.out.println("sleeping for 2 seconds...") ;
try
{
Thread.sleep(2000) ;
}
catch(InterruptedException e2)
{
System.out.println(e2.toString()) ;
}

System.out.println("waking up...") ;
java.io.OutputStreamWriter(s.getOutputStream())) ;
PrintStream out = new PrintStream(s.getOutputStream());
out.println(companyname) ;
out.flush() ;
out.println(addr1) ;
out.flush() ;
out.println("end of transmission") ;
out.flush() ;
System.out.println("finished writing data") ;

     }
     catch (IOException e) {
        showStatus("Server not activated: " + e);
     }
     finally {
        if (s!=null) try { s.close(); } catch (IOException e) {}
     }



  }


  public boolean action(Event e, Object o)
  {
     if(e.target == submit) {
      System.out.println("I'm in submit") ;
        writeMember();
        System.out.println("I just submitted") ;
        return true;
     }
    return false;
  }


}
```

Figure 8. Client code

```
import java.awt.*;
import java.applet.*;
import java.io.*;
import java.net.*;

public class ProtoServer extends Thread
{
  static final int defPort = 8081;
  static Socket s;
  static int counter = 0;
  boolean tFinished;
  static Thread sThread;

static void serverRequest(Socket ss)
{
boolean tFinished = false;
counter = counter + 1;
s=ss;
Thread thread;
ProtoServer sServer = new ProtoServer();
thread = new Thread(sServer);
sThread = thread;
thread.start();

}

public void run()
{
try
{
String line,line2;
File file = new File("stub.txt") ;
RandomAccessFile in = new RandomAccessFile(file, "r") ;
java.io.OutputStreamWriter(s.getOutputStream())) ;
PrintStream out = new PrintStream(s.getOutputStream()) ;
//Signal that a connection has been established.
System.out.println("connection established") ;
//Read a file and send text to client
while(in.getFilePointer() < in.length())
{
out.println(in.readLine()) ;
out.flush() ;
}
out.println("end of transmission") ;
out.flush() ;

//Get text from client and append to a file
DataInputStream in2 = new DataInputStream(new
BufferedInputStream(s.getInputStream())) ;
File file2 = new File("mem" + counter + ".txt") ;
RandomAccessFile out2 = new RandomAccessFile(file2, "rw") ;
System.out.println("about to enter second while loop to write a file")  ;
line2 = in2.readLine().trim();
while(!line2.equals("end of transmission"))
{
System.out.println(line2) ;
out2.writeBytes(line2 + "\n") ;
line2 = in2.readLine().trim();
}


in.close() ;
out.close() ;
out2.close() ;
```

(continued on following page)

Figure 9. Server code

(continued from previous page)

```
in2.close() ;
System.out.println("files closed");
//stop thread from running and free up resources
stopThread(sThread);

} catch(IOException w) {}

}

public void main(String args())
{
boolean running = true;
int thePort;
ServerSocket sSocket;

if (args.length == 1)
    thePort = new Integer(args()).intValue();
else
    thePort = defPort;

System.out.println(thePort) ;

     try {
       sSocket = new ServerSocket(thePort);
     }
       catch (IOException e) {
       System.err.println("I/O Server exception 1: " + error);
       return;
     }

     System.err.println("listening on " + thePort);
     while(running) {
       Socket soc = null;
       try {
       soc = sSocket.accept();

     System.out.println("it is accepted") ;
     System.out.println("about to call serverRequest") ;
     serverRequest(soc);
     System.out.println("handled the request") ;
       }
     catch(IOException e) {
       System.err.println("I/O Server exception 2: " + error);
       }

     }
   }

   public void stopThread(Thread ssThread)
   {
     if (ssThread != null)
       {
         ssThread.stop();
         ssThread.destroy();
       }

   }



}
```

Figure 9. Server code

```
<HTML>
<BODY>
<APPLET
  CODE=ProtoClient.class
  WIDTH=400
  HEIGHT=200>
</APPLET>
</BODY>
</HTML>
```

Figure 10. HTML code

Figure 11 shows the client registration form.

Figure 11. Registration form from client

Conclusion

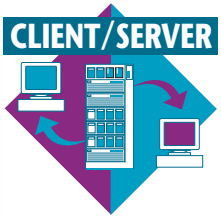
This article discusses a stand-alone server and a client applet portion of a Java client/server prototype. Fundamental client/server concepts and techniques are illustrated including file I/O, client and server connections, port requests, reading/writing to sockets, "cleaning-up" by closing all I/O streams, and handling client-side transactions via multithreading. It is easy to quickly prototype "compact" Web applications because Java provides easy socket programming via `Socket` and `ServerSocket` classes. It is also very functional regarding input and output data streams.

Acknowledgments

The author would like to thank Mickey Nix for technical assistance with helping to debug part of this code, and George Noren for good Java moral support!

Peter Farrett, *IBM Corporation, 11400 Burnet Road, Austin, TX 78758. Dr. Farrett is the technology leader for the IBM Solution Developer Program's Web site (www.developer.ibm.com) with responsibilities for developing new Web technology applications for Java, chat rooms, multimedia, and assessing future technological directions. He has an MSc and PhD in Computer Science from The City University of New York and Queen's University in Ontario, Canada, respectively.*

Data Parallel Programming with HPF on the RS/6000 SP



By David Klepacki and Xianneng Shen

This article is the third in a series about parallel programming models on the RS/6000 SP. The focus is on the features of the data parallel model, which is most often employed using the High Performance Fortran (HPF) Specification.

The first article[1] in this series briefly introduced the IBM RS/6000 SP™ architecture and its parallel programming models. Previous articles[2,3] discussed the message-passing model and its standard interface (MPI). This article describes an alternative to message passing: the data parallel model. The industry-standard specifications for this type of model are known as High Performance Fortran (HPF) and the more recent Data Parallel C (DPC). This article discusses the HPF implementation of the data parallel model.

The data parallel model is not new. It was predominantly used in the late 1980s during the era of parallel computers with the single instruction-multiple data (SIMD) architecture. SIMD characterizes how computations were carried out on this type of machine. In essence, each parallel processor would execute the same instruction simultaneously, but on different data. For example, a large two-dimensional matrix could be broken down into a "checkerboard" of smaller submatrices. Different processors could then operate upon these submatrices in parallel. This data parallel model could easily and naturally accommodate many engineering and scientific applications.

The scientific/engineering community quickly realized the need for a standard language/interface specification, especially for Fortran. The result was the HPF 1.0 specification, which appeared in 1992.[4] During that same period, rapid changes occurred in technology, which resulted in the demise of the SIMD architected computers in favor of more cost-effective and general-purpose computers with multiple instructions-multiple data (MIMD) architecture, such as the IBM RS/6000 SP.

MIMD enables multiple processors to operate on different instructions and different data at the same time. However, as described in the first article of this series,[1] the programming model is independent of the underlying hardware architecture.

[1] Klepacki, David and Shen, Xianneng. "Parallel Programming Models on the IBM RS/6000 SP." *AIXpert* magazine, September 1997.
[2] Shen, Xianneng; Ho, Eddie; and Hammill, Mike. "Message Passing Interface for RS/6000 SP." *AIXpert* magazine, March 1997.
[3] Shen, Xianneng and Klepacki, David. "Message Passing on the RS/6000 SP." *AIXpert* magazine, December 1997.
[4] High Performance Fortran Forum, "High Performance Fortran Language Specification, Version 1.0", *Technical Report CRPC-TR92225*, Center for Research on Parallel Computation, Rice University, Houston, TX, 1992.

The data parallel model with HPF is as applicable on today's MIMD machines as it was on the older SIMD machines. In fact, three HPF compilers are available today for use on the IBM RS/6000 SP: xlHPF (IBM), pgHPF (Portland Group Inc.), and xHPF (Applied Parallel Research Inc.). The HPF Specification is now in its second revision (HPF 2.0, 1996).

High Performance Fortran

HPF is a set of compiler directives that facilitate data parallel programming. Although an in-depth discussion of HPF is beyond the scope of this article, its fundamental components consist of the following:

- Data partitioning directives, such as `TEMPLATE`, `ALIGN`, `PROCESSOR`, and `DISTRIBUTE`
- Data parallel execution features, such as the `FORALL` statement
- Extended intrinsic functions; for example, `number_of_processors()`
- Extrinsic procedures; Sequence and Storage Association

HPF allows the programmer a high-level expression of parallelism. In the message-passing model, the array elements of distributed data derive new offsets relative to the local name spaces of each process. HPF is based upon a global name space, where the variables and arrays are manipulated from a global perspective (as in a serial application).

HPF programmers do not have to deal with the lower level nuances, such as buffering and synchronization, as in the message-passing programming model. HPF programmers need only be concerned with the distribution of data among the processors. The HPF environment performs the necessary low-level operations in order for this to occur. Therefore, the programmer can concentrate on the physical/numerical solution aspect of the application at hand.

On the RS/6000 SP, the HPF environment, not the user, transparently manages the physical message passing between the processors. Therefore, message passing can be viewed as the "assembly language" of parallel programming. The data parallel model provides a high-level language interface to parallel programming. However, this typically does not come without a cost. The ease-of-use associated with HPF usually results in sacrificing some degree of performance, not unlike the classical difference between high-level language programming and assembly language programming.

Fundamentals of HPF

Data parallel programming generally consists of three basic steps:

1. Create logical templates for the data and decide how the data should be partitioned. This should exploit the natural parallelism inherent in the application, not the computer.

2. Define virtual topologies for the processors (for example, a two-dimensional mesh or a three-dimensional cube). Typically, these topologies mimic the physical geometries and exploit the natural symmetries of the application itself.

3. Map the templates onto the virtual processor topologies. The HPF environment will then provide the necessary translation of operations onto the physical processors during compilation and runtime.

Let us consider how these steps are implemented in HPF using a simple example of rank sorting a vector of numbers. In Fortran 77, the serial program to perform this function is shown in Figure 1.

```
subroutine permute (key, rank, temp, n)
integer n, key(n), rank(n), temp(n)
do i = 1, n
    temp(i) = key(rank(i))
enddo
do i = 1, n
    key(i) = temp(i)
enddo
```

Figure 1. Rank sorting of a vector of numbers

Three vectors of length n are passed into this subroutine. The key vector is reordered according to the values in the rank vector. Because of the indirect addressing involved, it is necessary to introduce a temporary vector to hold the values of the keys as they are being reordered. This keeps key values that have not yet been reordered from being overwritten. Lastly, the key vector is updated from the temporary vector.

Parallelization with HPF

We want to parallelize this subroutine using HPF. The first step is to identify the logical structures to be partitioned, then determine exactly how they should be partitioned. In this example, all structures are in the same form: a vector of length n. So, our logical structure of partitioning can be represented as t(n). HPF has three methods of partitioning data:

- BLOCK
- CYCLIC
- BLOCK-CYCLIC

The BLOCK scheme partitions a vector into contiguous sub-blocks, each of length n/p, where p represents the number of processors available for this computation. The CYCLIC scheme assigns each consecutive element of the vector to a different processor, "round-robin" style (for example, as in dealing cards). The BLOCK-CYCLIC scheme is a combination of the previous two schemes. It assigns sub-blocks that are much smaller than the n/p size in the BLOCK case to processors round-robin style. We chose the BLOCK scheme for our example.

The second step is to define a virtual processor topology. This step is optional on the RS/6000 SP, and the default topology is a set of processors that are completely connected. However, for illustration purposes, we chose a linear array of processors p to mimic the vector nature of our application.

Lastly, we map our variables onto our virtual processor grid using the distribute directive. The final parallel subroutine is shown in Figure 2.

The template directive defines the fundamental logical structure of a partitioned

```
 subroutine permute (key, rank, temp, n)
 integer n, key(n), rank(n), temp(n)
!hpf$ template t(n)
!hpf$ align with t :: key, rank, temp
!hpf$ processors p(number_of_processors())
!hpf$ distribute t(BLOCK) onto p
 do i = 1, n
    temp(i) = key(rank(i))
 enddo
 do i = 1, n
    key(i) = temp(i)
 enddo
```

Figure 2. Final parallel program

object. Since it does not allocate any storage, there is no need to declare its type in the program. The align directive is optional here, but we use it to illustrate good programming practice. Typically, the arrays and vectors of more complex programs will require alignment due to variations in their declared sizes.

The processor topology is defined with the processor directive. In this example, it uses the HPF Extended Intrinsic function, number_of_processors(), which returns the number of physical processors employed at runtime. Using this function, the programmer does not need to hard-code the program for any specific number of processors. Rather, this program will function on any number of processors without the need to recompile each time a different number of them are used.

This application has been parallelized with the addition of four comment-based directives; therefore, this code still runs unchanged on a stand-alone workstation. It is simple to change the distribution of the data among the processors and/or the processor topology. For example, cyclic data distribution among the processors requires only that the BLOCK parameter be changed to CYCLIC. As another example, introducing a three-dimensional grid of 27 processors can be obtained with the specification of q(3,3,3).

More than one processor topology can be defined concurrently. More complex applications may require one topology during one part of the computation and another topology during another part of

the computation. The reader should compare and contrast these abilities with the message-passing model. In general, these data parallel constructs are not as easily handled with the message-passing model.

FORALL Construct

The `FORALL` statement is another powerful feature of HPF. It provides a convenient syntax for simultaneous assignments to array elements. Figure 3 shows some examples.

Note that the parallelism is implied. The arrays can be physically partitioned via the `distribute` directive. All of the necessary interprocessor communication is transparently performed by the HDF environment.

Summary

Data parallel programming is a viable and powerful model when used appropriately. In particular, it should be used whenever regular structures of data are manipulated in a static and predetermined way. Otherwise, load-balancing issues arise, which would be better handled by one of the other two programming models.

To really appreciate the power of HPF, you should attempt to parallelize the simple rank-sort programming example in this article using the message-passing model (for example, `MPI`). This example will be reviewed again in the next article of this series, which will discuss the Virtual Shared Memory model.

Acknowledgments

The authors would like to thank the following people for invaluable discussions on HPF: Christopher Kerr (IBM Corporation), John Levesque (Applied Parallel Research, Inc.), and Doug Miles (Portland Group, Inc.).


```
! A = Transpose of B
FORALL (i=1:m, j=1:n) A(i,j) = B(j,i)

! Assignment to diagonal elements
FORALL (i=1:n) A(i,i) = B(i)

! A of the second assignment uses values
! of the first assignment
FORALL (i=2:n-1, j=2:n-1)
  A(i,j) = A(i,j-1) + A(i-1,j)
  B(i,j) = A(i,j)
END FORALL
```

Figure 3. FORALL statement examples

***Xianneng Shen**, IBM Corporation, 522 South Road, Poughkeepsie, NY 12601. E-mail: xshen@us.ibm.com. Dr. Shen is a programming consultant in the RS/6000 Executive Briefing Center. He has a BS and an MS in Electrical Engineering from the University of Electronic Science and Technology of China, an MS in Computer Engineering from Syracuse University, and a PhD in Electrical Engineering from Syracuse University.*

***David Klepacki**, IBM Corporation, 522 South Road, Poughkeepsie, NY 12601. Dr. Klepacki has been working in IBM's POWERparallel Systems Group since its beginning in 1991 as a computational physicist and scientific applications specialist with emphasis on performance benchmarking. Today, in addition to his technical endeavors, he also manages the parallel software tools segment for the technical marketing branch of the RS/6000 Division. Dr. Klepacki's current interests include performance programming, scalable parallel algorithms, scalable I/O, and portable high-performance computing tools. He holds a PhD in Theoretical Nuclear Physics from Purdue University as well as an MS in Electrical Engineering from Syracuse University.*

Network Station S/1000: A New Java Device

By Eddie Ho, Steve Heracleous, and Ravi Mandava

The IBM Network Station Series 1000 extends the IBM Network Station family of network computers by delivering robust support for business applications written in Java while effectively meeting other desktop computing demands, from accessing traditional business applications to accessing the corporate intranet and the Internet.

Like all IBM Network Stations, Series 1000 offers the power of network computing to everyone throughout the enterprise. Most important, the Series 1000 supports the Java Virtual Machine (JVM) 1.1.2, thereby delivering the full power of Java™ applications. This JVM level supports the classes and execution environment for Java applications. Over time, a new version will be upgraded to provide a richer level of functions.

With the Network Station™ Series 1000, all data and applications—as well as configuration, maintenance, and troubleshooting—reside on the server. This results in a computing environment that is easier, safer, and less expensive to set up and maintain. In addition, the Series 1000 significantly enhances the enterprise's ability to do all the desktop-level computing it needs, using the cost-effective power of the network.

The Java-focused Network Computer

The IBM Network Station Series 1000 is the Java-focused network computer. It is designed with adequate compute power to run applications locally. The previous models (100 and 300) are designed to access server-based applications with limited Java and network bandwidth capability. Figure 1 shows the Series 1000 at a glance.

The IBM Network Station Series 1000 is optimized for running business-critical applications and personal productivity tools that take advantage of Java. It also accommodates extensive use of corporate intranets or the Internet. The IBM Network Station lets you do the following:

- Run Java applets and applications directly on the Network Station. The industry is pursuing an architecturally neutral programming model using the Java language and platform technology. Eventually, the Internet will become the universal server for all types of content and applications. The Network Station is designed to execute applets on demand with the highest level of performance.

- Access Windows™ applications via multi-user implementations of Windows NT™ on a PC server. The

Network Station is also an efficient X protocol server capable of displaying Windows NT applications in a multi-user environment. The WinCenter Pro environment enables graphics and audio to be exported.

- Use a browser to access multiple servers (IBM and others), including corporate intranet and Internet servers. The Network Station has a locally executed browser and is functionally equivalent to the Netscape Navigator™ 3.0 level of function. As the business environment makes the transition to Internet-/intranet-centric applications, the browser is the essential converged user interface for all Web-enabled applications.

- Access 3270 and 5250 terminal applications. The Network Station supports the 3270 and 5250 client for convenient access to enterprise-based applications. This access is critical to preserve the existing production environment while making the transition to a more Web-centric model.

- Work with applications on AIX® and UNIX® servers using X-Windows server support. The Network Station can optimize networked resources by combining the processing power of the servers and the Series 1000.

All-in-One Application Access Device

With the Network Station, traditional business applications are available where and when you need them. They also have a new user-friendly, productivity-enhancing graphical front-end. This graphical window interface allows concurrent access to existing server-based applications or Web-enabled applications.

A new, highly graphical commercial or personal productivity application, such as Lotus eSuite WorkPlace™ (from Lotus Development Corporation) written in 100% Pure Java language, will run on the IBM Network Station Series 1000, along with other familiar platform-specific applications from OS/390™, AS/400®, UNIX, and Windows NT environments. Figure 2 shows connectivity using the Network Station. See the sidebar for more information about Lotus eSuite Workplace.

Network Station Series 1000 at a Glance

Models	Ethernet, Token-Ring (1Q98)
Terminal support	3270, 5250, X-Windows server
Java Virtual Machine	1.1.2
Web browser	Supported
Windows applications	Via multi-user implementations of Windows NT on a PC Server[1]
Lotus Notes®	Via Web browser and Domino server
Memory	32 MB EDO (base) expandable to 64 MB, 2 SIMM Sockets, Optional 512 KB SRAM cache memory
Connectivity	Ethernet 10/100 Mb or Token Ring 4/16 Mb
I/O ports	One serial, one parallel
Video support	Minimum: 640x480 VGA Maximum: 1600x1280 SXGA 2 MB (base) VRAM
Monitor support	Video graphics array (VGA) Super video graphics array (SVGA) Super extended graphics array (SXGA)
Smart card support	Hardware support only
Input devices	102-key PC keyboard, two-button mouse
Audio support	16-bit audio

[1] *Such as WinCenter available from Network Computing Devices, Inc.*

Figure 1. Network Station Series 1000 at a Glance

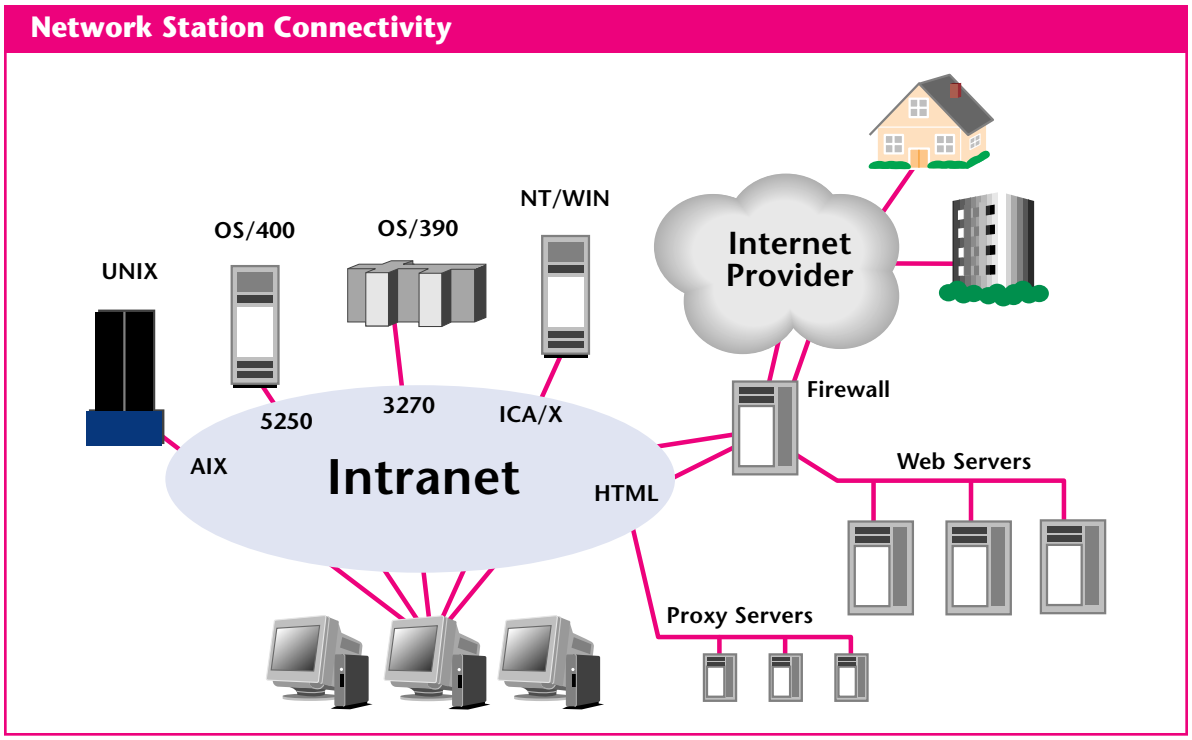


Figure 2. Connectivity of the Network Station

Desktop Consolidation Using Network Station

The IBM Network Station Series 1000 is an ideal desktop solution for the following:

- Deploying new Java-based business applications. You can develop a Java application on a familiar platform. The Network Station can then provide the most optimized execution environment if the virtual machine has the same level and is 100% pure.
- Activating existing business applications with user-compelling graphical front-ends. The Network Station can modernize the dull green screen-based application, thereby improving your productivity with a higher level of window concurrency.
- Moving to Internet and intranet architectures that still need access to business applications on multiple servers. The Network Station can provide a single desktop consolidation strategy and eliminate the need for multiple types of terminals.
- Accessing applications delivered through corporate extranets (e.g., order tracking, package tracking, human resources).
- Providing concurrent access to multiple platforms. In a traditional Web application development environment, concurrent access to multiple platforms is a basic requirement. Java application developers often need access to multiple servers and the Internet. The Network Station allows developers to code and debug on the server's platform, then conduct a production test via the browser. If the design is download-on-demand, the local Java runtime is already tuned and optimized.
- Storing vital data centrally rather than on local PCs. This is important to security-conscious organizations, such as financial institutions, insurance companies, and government agencies.

- Replacing inefficient or difficult-to-maintain older terminals and/or older PCs or adding new terminals or PCs.
- Consolidating PCs—running PC applications on the server to cut desktop-level maintenance and troubleshooting costs.
- Increasing information technology (IT) functions at a low cost.

Java Virtual Machine Environment

The Network Station Series 1000, unlike its predecessors, can run the Java Virtual Machine Version 1.1. This version offers many new features. It also provides functional fixes for defects and enhancements as

Introducing the Lotus eSuite WorkPlace

The new Lotus eSuite WorkPlace provides a whole new way of looking at network computing and Java. The WebTop productivity technology is another level of integration for a total Java desktop. Current PC space has very efficient desktop productivity tools; however, they are all OS specific and can be accessed using a Network Station in a multi-user environment. A total Java desktop running on the Network Station enhances the office productivity of end-users, but still allows centralized control of all contents. Java-based applications developed in other platforms can be easily integrated in the desktop.

Lotus eSuite WorkPlace, a new class of productivity software, is designed specifically for the emerging network computing environment. The eSuite WorkPlace provides a single point of access to everything most computer users need, including business productivity applets, integration with existing applications, terminal emulation, e-mail, calendaring and scheduling, and the Internet.

Lotus eSuite WorkPlace allows anyone in an organization to access the Web, manage documents, and perform tasks crucial to their work. IT managers and network administrators can customize the eSuite WorkPlace. When users need additional capabilities, simply supplement eSuite WorkPlace with the appropriate Java applet—including applications built with the eSuite DevPack, or any third-party Java applet or application.

Included with the eSuite WorkPlace is a set of eSuite business productivity applets that provide the basic functions users need to work more productively, without complicated "extras." The eSuite applets are 100% Pure Java, so they cross all computing platforms. This allows IT managers to deploy eSuite applets quickly across the enterprise, yet maintain and upgrade them centrally.

Lotus eSuite WorkPlace includes:

- Desktop user interface, including a Web browser, terminal emulation, and file manager
- Business productivity components, including a word processor, spreadsheet, presentation graphics, calendar, address book, e-mail, project scheduler

Lotus eSuite components share a common, easy-to-use interface called the InfoCenter. This is where you can set attributes and access the functionality of any eSuite applet to enable users to get up and running quickly.

Lotus eSuite WorkPlace is the ideal solution for PC users who want an easier, more flexible, productive working environment. At the same time, eSuite WorkPlace provides terminal users with access to a whole new range of desktop tools and capabilities, along with improved access to existing host-based business applications.

For the latest information about Lotus eSuite, visit the World Wide Web at *http://eSuite.lotus.com*.

well as performance improvements. The highlights of Version 1.1 consist of the following:

AWT enhancements. JVM Version 1.1 offers many enhancements compared to Version 1.0.2. These enhancements, which solve Abstract Window Toolkit (AWT) deficiencies, include application programming interfaces (APIs) for printing, easier and faster scrolling, pop-up menus, clipboard (for copy and paste), user-defined cursors per component, a delegation-based event model, imaging and graphics enhancements, and more flexible font support for internationalization.

JavaBeans. The JavaBeans™ API defines a software component model for Java, allowing third-party ISVs to create and ship Java components that end users can combine together into applications.

JAR file format. The Java Archive (JAR) is a platform-independent file format that allows Java developers to bundle multiple class files, graphics, and sound files into one file that can subsequently be downloaded in a single HTTP transaction. This greatly enhances the download speed of a set of classes and support files required to execute an applet since multiple HTTP transactions (one per class file) are now avoided. In addition, the resulting JAR file is compressed, which further enhances the download speed.

Security and signed applets. JVM 1.1 provides a tool that allows developers to assign a JAR file, which typically contains class files, image, and sound files. The appletviewer can then allow a trusted entity to run any downloaded applets in signed JAR files with the same privileges as a Java application would have on the client platform (accessing local storage).

Furthermore, the Java Security API provided with the Java Development Kit (JDK) 1.1 enables developers to incorporate security functionality into their Java applications. The first release of Java Security in JDK 1.1 contains a subset of this functionality, including APIs for digital signatures and message digests.

RMI/object serialization. Remote method invocation (RMI) enables developers to create distributed Java applications. Methods of remote Java objects can be invoked from other Java virtual machines residing on different hosts. Using a naming service provided by RMI, a Java object can call a method of a remote Java object. RMI uses object serialization to marshal and unmarshal parameters to the remote object. Object serialization allows encoding of objects into a stream of data bytes (which can be directed to persistent storage) and the reconstruction of the object from such a stream of data bytes. It also allows for lightweight object persistence.

Reflection. Reflection enables a Java program to examine the class definition of an object at runtime. It also supports the extraction of the full definition of the accessible APIs for the `object=s` class, which lists the constructors, methods, and variables. Accessing variables and invoking the methods are supported.

Java DataBase Connectivity. Java Database Connectivity (JDBC) provides a standard SQL interface for uniform access to a wide range of relational databases. Using JDBC, developers can build higher level interfaces, tools, and access front-ends to relational databases.

Other Enhancements. JVM 1.1 provides other numerous enhancements, including:

- Internationalization allows applets and applications to be developed using a local mechanism, localized message support, and display of Unicode characters.
- Networking enhancements are available in the `java.net` base classes.

- I/O enhancements allow character streams based on 16-bit Unicode characters rather than 8-bit bytes.
- Inner classes have an interface by which classes can define other classes/interfaces as part of their definition. This new language feature applies to Java the concepts of lexical scoping and block structure found in many other programming languages, but not available in Java 1.0.
- Java Native Interface (JNI) has a mechanism by which Java can reference native (to a particular platform) libraries.

Conclusion

The Network Station Series 1000 with its integrated desktop can consolidate all your desktop devices into one. It can also bridge your computing environment to the future with the promise of a platform-neutral execution environment.

Eddie Ho, *IBM Corporation, 11400 Burnet Road, Austin, TX 78758. Mr. Ho is a programming consultant in the Technical Marketing Center, Network Computer Division. He has a BS in Computer Science from University of Wisconsin and an MS in Computer Science from North Dakota State University.*

Steve Heracleous, *IBM Corporation, 11400 Burnet Road, Austin, TX 78758. Mr. Heracleous is an advisory programmer in the Technical Marketing Center, Network Computer Division. He has a BS and MS in Computer Engineering from the University of Texas in Austin.*

Ravi Mandava, *IBM Corporation, 11400 Burnet Road, Austin, TX 78758. Mr. Mandava is an advisory programmer in the Network Computer Division Development. He has a BS in Mechanical Engineering and a MS in Computer Science from Osmania University in India.*

Avoiding CGI

By George Kraft IV

Many Internet Service Providers do not allow their customers to publish their own interactive graphic image maps on the Web, query the reader, or keep track of reader statistics. The average Web author must work around the lack of Webmaster privileges on the server.

The Internet and the World Wide Web have become an important tool for both large and small businesses. They have also become increasingly entertaining for individuals. Both commercial enterprises and individuals write and publish a vast amount of information on the Web. Large corporations publish information about their products and services on the Web; they also can receive consumer orders and information from around the globe. Individuals publish information about their hobbies, interests, and academic endeavors; however, they are rarely able to create interactive graphic image maps, query the reader, or track the number of readers who have visited their Web site.

The Common Gateway Interface (CGI) controls much of the World Wide Web's interactive capabilities. The CGI suite of programs reside on a Web server, such as an RS/6000™, that takes input from HyperText Markup Language (HTML) document browsers and returns new dynamically created HTML information.

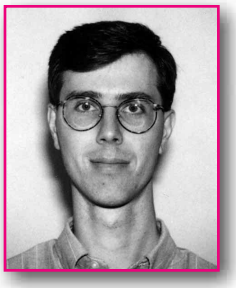
A problem exists, however, because there is no standard set of CGI scripts for all of the different Web server software suites available. Publishers must use what is available on the server or write new scripts; this is not possible for most domestic Web publishers. Most individuals rent Web space for their pages from Internet Service Providers (ISPs). Generally, ISPs are unable and/or unwilling to give access to their subscribers to create interactive CGI scripts. They are concerned about the security implications of these CGI scripts and also the impact of these scripts on the system because of the pervasive global access.

To write interactive graphic image maps, query the reader, or track the number of readers who have visited your site, you can use what the ISP might be providing or you can write HTML pages that do not require the use of CGI.

Image Maps

HTML image maps enable graphical images to have hyperlink capabilities. Users can move the pointer over an image and select certain areas that result in loading a new HTML page. Originally, CGI scripts processed image maps, which meant that images and their maps were sometimes required to be installed within the Web server and could not be maintained by an HTML author. This process allowed only the Webmaster to maintain image maps.

Today, most state-of-the-art Web browsers allow client-processed (that is, browser-processed) image maps by using the new HTML directive called *MAP*. MAP



George Kraft IV

allows HTML authors to define the image's hyper-reference regions within a normal HTML file. Now it is not necessary for a separate graphics image, image map, and CGI program to dynamically process the information. Today, the browser processes all image map information.

Figure 1 shows an example of a client-side (browser) image map. The IMG image is defined to use the source GIF file `rgb.gif`, and it is extended to use the image map labeled `RGB_MAP` in Figure 2. The image map defines various regions and coordinates that are hyper-reference links to other Web pages. The user only needs to point and click on the image-mapped graphics in order to follow the hyperlink. Since the browser processes everything locally, there is no dependency on the remote Web server.

When the user moves the pointer around the graphics image, the browser's status bar changes to the alternative text (`ALT`) of the defined region. If the user clicks in that region, then the hyper-referenced page linked to that area is loaded in the browser. All of the image map processing is done locally in the browser.

Image Map Example

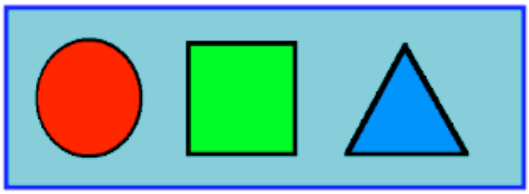


Figure 1. Example of an image map

Forms

HTML forms can prompt the user for useful textual input. Usually an HTML form prompts the user for text, a selection from multiple choices, or a series of true or false questions. Then, the user selects the "submit" button and the browser sends the HTML-prompted information to a CGI-hosted program as input. The information is processed and a resulting HTML output is sent back to the browser to be dynamically loaded as a response. Figure 3 shows an example of an e-mail form.

Normally, Webmasters set up forms and a local depository for information taken from the Web; however, most domestic Web publishers hosted by ISPs do not have access to save the data taken from readers on the Web. Instead, Web authors have the option to e-mail the user's response via the

E-mail Form Example

Figure 3. Example of an e-mail form

```
<CENTER>
<IMG SRC="images/RGB.gif" ALT="Red Green Blue" USEMAP="#RGB_MAP">
<MAP NAME="RGB_MAP">
<AREA shape=circle coords="55,55,50"  ALT="Red"   HREF="red.html">
<AREA shape=circle coords="145,55,50" ALT="Green" HREF="green.html">
<AREA shape=circle coords="240,60,50" ALT="Blue"  HREF="blue.html">
</MAP>
</CENTER>
```

Figure 2. Image map HTML code

form's ACTION. E-mailed information that is prompted from HTML forms can be batch or dynamically processed, but the e-mailed forms processed by e-mail cannot give a browser any kind of dynamic response. After the browser e-mails the information, it is processed by an entirely different method of filtering. Figure 4 shows HTML code for an e-mail form.

On UNIX®, users can set up filters to process batch information sent by e-mail. The `sendmail` service on UNIX systems processes all e-mail through the users `$HOME/.forward` file. The `.forward` file can either redirect the e-mail to another location or it can pipe the information through programs to be processed. The Mail Handler (MH) and Elm/Pine e-mail readers allow information to be processed before the messages are delivered into the user's inbox. E-mail filtering is an opportunity to process HTML forms without relying on the ISP's CGI scripts.

Counters

After authors publish their Web pages, they sometimes find it useful to know if and how many readers have visited selected materials from their Web site. ISPs may or may not provide a Web page counting service. This is unfortunate because individual browsers cannot keep track of global access of a Web page. The counting of Web pages must be stored somewhere on the Internet, but not necessarily on the server where the Web site resides. The Internet has many free Web-page counting services available. But the author's HTML page normally must include a small advertisement banner to pay for the service, as shown in Figure 5.

```
<FORM METHOD=post ACTION="mailto:gk4@ibm.com">
<TEXTAREA name=inquiry rows=8 cols=40 wrap=physical>
This area could be prefilled with text.
</TEXTAREA>
<BR>
<INPUT type=submit value="EMail">
<INPUT type=reset value="Reset">
</FORM>
```

Figure 4. E-mail form HTML code

Figure 6 shows the HTML encoding for a counter, which includes a CGI counter with the author's account and a hyper-referenced GIF advertisement. When the browser loads the Web page, the CGI counter accesses the authors account, the index is incremented, and a returning image is returned with the page's current access count. If the user clicks on the banner, then the advertiser's page is loaded, thereby paying for the service of the counter.

Figure 5. An example of a Web page counter

```
<!——-Begin code for Free counter——————————————>
<TABLE border="0" cellpadding="0" cellspacing="0" width="300">
<TR><TD><CENTER>
      <IMG SRC="http://www.free.com/cgi-bin/pagecount?account=kraft">
</CENTER></TD></TR>
<TR><TD>
      <A HREF="http://www.marketing.com/advertisement.html">
            <IMG SRC=”advertisement.gif">
      </A>
</TD></TR>
</TABLE>
<!——-End of free counter code————————————————>
```

Figure 6. Web page counter HTML code

Conclusion

As more HyperText Markup Language editors become available and Internet Service Providers provide less expensive Web space, it is easy to establish a presence on the World Wide Web. You can author Web pages using editors like Microsoft's FrontPage98 or NetObjects' Fusion™, but be aware that you may not have complete access to all of the features that HTML authoring has to offer. Be careful when using CGI scripts that are ISP dependent, so make a conscious decision when coding special features.

***George Kraft IV**, IBM Corporation, 11400 Burnet Road, Austin, TX 78758. Mr. Kraft is an advisory software engineer for IBM's new Network Computer Division. He recently moved from IBM's RS/6000 Division where he worked on the AIX integration of the IBM Network Station. He has a BS in Computer Science and Mathematics from Purdue University.*

AIX Questions

Compiled by Jeff Simon

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 do I configure an anonymous ftp site in AIX?

AIX® provides a sample script to (minimally) configure the RISC System/6000® as an anonymous ftp server. You can find the script in `/usr/lpp/tcpip/samples/anon.ftp` (AIX 3.2.5) or `/usr/samples/tcpip/anon.ftp` (AIX 4.x). Follow the steps below to configure the RS/6000™ as an anonymous ftp server.

1. Execute the `anon.ftp` script, which will create the ftp and anonymous users. These users will have different User IDs (UIDs) but the same home directory. In addition, the script will create five subdirectories (`etc`, `pub`, `bin`, `lib`, and `usr`) under the ftp user's home directory, each owned by root. This prevents anonymous ftp users from making any changes. The `etc`, `bin`, `lib`, and `usr` directories are designed for ftp use; therefore, they have world-readable (`r`) and world-searchable (`x`) permission only. The `pub` directory also has world-writeable (`w`) permission to allow file uploads. Executing this script is sufficient to provide a minimal ftp server.

   Now, when a user ftps to the RS/6000 and specifies the anonymous or ftp user name, the ftp daemon automatically accepts any password. However, the ftp daemon also automatically executes the `chroot` command on the ftp user's home directory. This restricts an anonymous ftp user to the ftp user's home directory structure. Thus, anonymous ftp users can only access (that is, `cd` and `ls`) a limited directory structure, not the entire directory structure of the machine. Although the `anon.ftp` script provides a minimal setup, you may wish to perform some additional configuration.

2. Remove the `~ftp/.profile` created by the system for the ftp user.

3. Create a `passwd` and `group` file in the ftp user's `etc` directory. This will allow anonymous ftp users to see names instead of numbers for file permissions. The `passwd` file should look like this (assuming 302 is the ftp user's UID):

```
ftp:*:302:1:anonymous ftp user:/u/ftp:/bin/false
root:*:0:0::/:/bin/false
```

   The group file should look like this:

```
system:*:0:
staff:*:1:
```

Then, execute the `chmod 400 *` command in the `~ftp/etc` directory.

4. Enable logging on the ftp daemon, which allows you to record the hostname and e-mail address (anonymous password) of anonymous ftp users, as well as which files they upload and download. To enable logging, do the following:

   - Execute the `smit inetdconf` command (as root).
   - Select Change/Show Characteristics of an `inetd` Subserver.
   - Select ftp.
   - Change the Service Program command-line `ARGUMENTS` field to read `ftpd -l`, then press Enter.
   - Edit the `/etc/syslog.conf` file and add the line `daemon.info /tmp/ftp.log`.
   - Execute the command `touch /tmp/ftp.log`.
   - Execute the command `kill -1 <syslogd pid>`, where `<syslogd pid>` is the Process ID (PID) of the `syslogd` daemon. This will cause the `syslogd` daemon to re-read its configuration file.

5. Create a separate directory for uploads, then execute the following commands:

   - `cd ~ftp`
   - `mkdir incoming`
   - `chmod 777 incoming` (to allow uploads)
   - `chmod 555 pub` (to protect your outgoing repository from uploads)

—Jeff Simon

Does Base Operating System (BOS) 4.2 have a mechanism that will automatically export global functions from my shared libraries without generating an export list?

Yes, this can be done by linking with the `-bexpall` flag. The `-bexpall` flag exports all global symbols, except imported symbols, unreferenced symbols defined in archive members, and symbols beginning with an underscore (_). Additional symbols can be exported by listing them in an export file.

By using this option, you can avoid using an export file. On the other hand, an export file provides explicit control over which symbols are exported. It allows you to use other global symbols within the shared object without worrying about conflict with names exported from other shared objects.

See InfoExplorer™ (or man pages) under `ld` (look under Options) for additional information about `-bexpall`.

Jeff Simon

—Jeff Simon

I built my database application on BOS 4.1.5 and get the following error messages when running on BOS 4.3:

```
Could not load program ...
Symbol kaio_rdwr in /usr/lib/libc.a is undefined
Symbol listio in /usr/lib/libc.a is undefined
Symbol acancel in /usr/lib/libc.a is undefined
Symbol iosuspend in /usr/lib/libc.a is undefined
Could not load library libc.a[aio.o]
Error was: Exec format error
```

What is causing this problem?

You need to make asynchronous I/O available on your BOS 4.3 system by doing the following (as root):

```
<smit/smitty> chgaio
 State to be configured at system restart
```

Note: You can use the tab key to toggle from defined to available. Next, you will need to reboot the system to initiate any changes.

—Jeff Simon

How can I tell which APARs were installed on my system?

You can do this by entering one of the following commands:

```
instfix -ik <APAR#>      // query individual APAR#
  i.e. instfix -ik IX71075
```

or

```
instfix -iv | grep IX    // query all APAR #'s
```

—Jeff Simon

What does firmware do on RS/6000 machines?

Firmware is the first code executed when the machine is powered-on. Firmware takes the system from a power-on state to a state where an operating system loader is in memory, ready for execution. Firmware is typically stored in read-only memory (ROM) or in programmable read-only memory (PROM).

The firmware on PCI-based RS/6000 systems performs the following steps:

- Initializes processor registers
- Initializes the memory controller
- Establishes an active RAM area
- Copies decompressed code to RAM
- Decompresses the compressed area into RAM
- Establishes execution environment (stacks, and so on)
- Initializes the console
- Displays graphics/logo on console
- Initializes individual subsystems
- Locates and loads the operating system's boot code

The PCI-based RS/6000 system's firmware also provides functions that were already incorporated in the firmware for the Micro Channel®-based RS/6000 systems. One of them is Power-On Self Test (POST). The POST checks basic hardware, such as processor, native I/O, and system memory, and makes a list of the working hardware it recognizes.

When the system is booted, the firmware obtains the boot device list maintained in NVRAM and tries to locate the first valid boot device that has a valid boot image on it (this is determined by reading the first 512 bytes on a device). When a valid boot image has been found, it is loaded into memory and firmware passes control to it. The firmware does not have to be aware of the type of code it loads—Software Read-Only Software (ROS) or the kernel of an operation system. For AIX, it is Software ROS.

Wade Carlin

—Wade Carlin

What hardware platforms does AIX Version 4 support and how does this affect boot images?

AIX Version 4 provides support for the following three RS/6000 hardware platforms:

- Micro Channel
- Symmetric Multiprocessor (SMP)
- RSPC (PCI-based)

This AIX boot image consists of the following components:

- AIX kernel
- RAM
- File system
- Base customized device information

The Software ROS loads the AIX boot image. To create platform-specific boot images for AIX, the software must first recognize the platform type that is specified in the IPL control block built by Software ROS.

The boot utility command, `bootinfo -T`, inspects the IPL control block and prints the platform type to the standard output. The command `bosboot -T` generates the boot image for the target platform. If the platform type is not specified, this command defaults to the current platform type on which the system runs.

—Wade Carlin

```
$ ls -al
drwxr-xr-x 2 brenda staff 8192 Aug 26 13:45 .
drwxr-x-- 9 brenda staff 2048 Aug 26 12:03 ..
-rw-r--r- 1 brenda staff  695 Aug 26 13:45 file1
-rwxr-xr-x 1 brenda staff  695 Aug 26 13:45 script.ksh
```

Figure 1. Permissions assigned per file/directory

Brenda Hagler

Nilesh Gandhi

How can I set the default permissions on files that I create to make them read-only?

When you create a file, the umask defines the read, write, and execute permissions for the file. To set the permissions on the files to be created so that the default is read-only for group and others, use the umask of 033. To see what umask you are currently using, issue the `umask -S` command.

If you put the umask in the `/etc/profile` file, users on your system will create files using that umask. They can define their own umask by putting the umask assignment in their `$HOME/.profile`. The umask in the users `.profile` will take precedence over any umask defined in `/etc/profile`. The command `ls -al` will display the permissions assigned per file or directory. Figure 1 shows an example.

Umask Number	Permissions
umask 002	Files and directories will be created WITHOUT write permission for others.
umask 077	Files and directories will be created WITHOUT read, write, or execute permission for group or others.
umask 027	Files and directories will be created WITHOUT write permission for group and WITHOUT read, write, or execute permissions for others.

Figure 2. The umask permissions

Reading the permissions from left to right, the first read, write, execute (`rwx`) permissions belong to the owner. In the above example, the owner is brenda. The second set of `rwx` permissions belong to the group staff in this case. The third set of permissions are for other users. Figure 2 shows examples of umask numbers and their permissions.

Note: The umask number defines which permissions should NOT be given.

For more information, see the `umask` command in the *AIX Commands Reference* (SC23-2639-03). "System Startup Files Overview" in InfoExplorer reviews the `/etc/profile` versus a user's `.profile`.

—Brenda Hagler

How can sed be used to insert a new line?

Figure 3 illustrates how `sed` can substitute a new line for any pattern, using either a line-feed or a character sequence.

Method One

```
% sh> sed 's/ /\ (press return)
> /g' <fileName>
```

Method Two

```
% sh> sed 's/ /\^J/g' <fileName>
i.e. ^J can be created while holding control key
     and pressing v and j keys simultaneously.
```

Figure 3. Inserting a new line using sed

—Nilesh Gandhi

Why do I get a core dump when trying to read 'sed -f <fileName>' from a command line?

This problem may occur if the input file contains more than 300 lines of text or if it has over 32,000 characters. When the buffer is overwritten, the system detects a memory fault and causes the `sed` program to core dump.

—Nilesh Gandhi

***Jeff Simon**, IBM Corporation, 11400 Burnet Road, Austin, TX 78758. Mr. Simon has worked in technical support since 1990 and currently works as a technical lead and systems administrator for a series of technical Web sites. He has a BS in Computer Science from Southwest Texas State University.*