TPF Systems
Technical Newsletter

http://www.ibm.com/tpf/news/newshd.htm

First Quarter 2004 Volume 11 Number 1

In this issue

"With the proper use of the powerful tools of IDS, SSL, UACC, and the built-in DoS attack prevention, the TPF system is an extremely uninviting place for intruders."

Intruder Alert
Evan Jennings, IBM TPF Development

TPF Systems Technical Newsletter
Vol. 11 No. 1

Editor:	Frank DiGiandomenico
Graphics Support:	Stephanie Daniels

Comments, Address Changes

This newsletter is provided free-of-charge through the IBM TPF Web site. IBM reserves the right to discontinue this service or change any of the policies regarding this service at any time.

To Fax your comments, use the following number:

+1-845-432-9788.

For inquires, please click here for e-mail.

To send comments electronically, use the following Internet e-mail addresses:

dijohn@us.ibm.com

Letters to the Editor and Articles Policy

Suggestions on topics and comments on articles are welcome from our readers. Letters or articles submitted for publication by readers may or may not be published at the sole discretion of IBM.

IBM Corporation
TPF Systems Development
Mail Station P923
2455 South Road
Poughkeepsie, NY 12601-5400
USA

This publication was produced in the United States. IBM may not offer the products, services or features discussed in this document in other countries and the information may be subject to change without notice. Consult your local IBM business contact for information on the products or services available in your area.

Trademarks

The following is the trademark information for this newsletter issue.

IBM, EOCF/2, MQSeries, OS/390, and WebSphere are registered trademarks of International Business Machines Corporation in the United States, other countries, or both.

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both

Other company, product, and service names may be trademarks or service marks of others. For additional trademarks see the TPF Trademarks list.

Announcing IBM TPF Web-Based Product Support

Jonathan Collins, IBM TPF Development

What is your first step when you are u nable to solve a TPF problem? Whether you are a new user looking for introductory information, or an experienced user looking for a solution to a problem, TPF Web-Based Product Support can help your research by providing access to a repository of information gathered from various technical resources. We recommend that you incorporate TPF Web-Based Product Support as part of your problem determination process.

TPF Web-Based Product Support is curre ntly available from the Support area of the TPF Web site at http://www.ibm.com/tpf/infohelp/index.htm. This interface helps you to identify information from a variety of sources, including the TPF Web site, the IBM eSupport database, and the TPF Product Information Centers.

The following describes how to start your search.

1. Under **Enter search terms**, enter the word or words that you are interested in finding.

2. Under **Select search domains**, select one or more of the following:

- **FAQs:** Frequently asked questions, primarily in the areas of:
  - MQSeries
  - Communications
  - Recoup
  - C/C++
  - TPF Operations Server
  - Program Update Tape (PUT) Migrations
- **Newsletters:** These are articles published on the TPF Web site ranging from in-depth information about the TPF family of products to announcements about new functionality. These articles usually are written by subject area experts with a great deal of insight relevant to the specific area of discussion.
- **Service Bulletins:** These are short announcements that inform you about important APARs or changes to how we support the TPF family of products. Service bulletins often point you toward other sources for additional information about a particular subject.
- **TPF Users Group Presentations:** These are a collection of presentations given by IBM at TPF Users Group conferences.

3. Click **Go**.

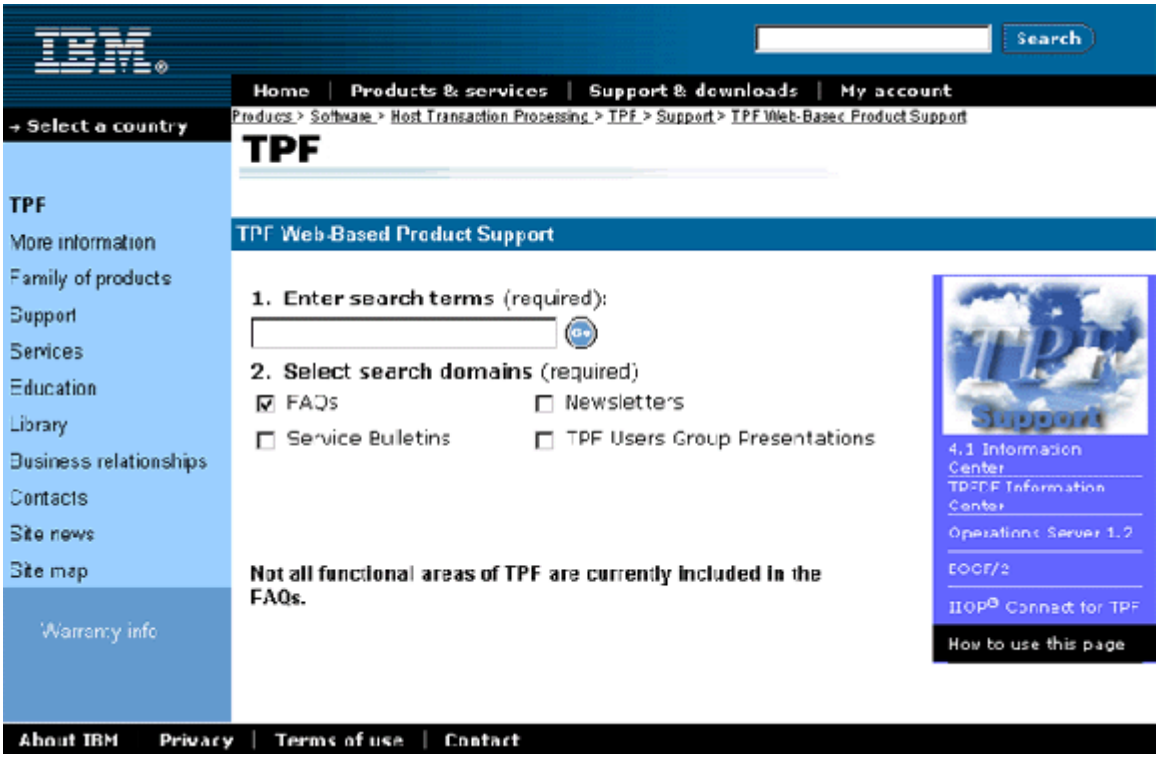


Figure 1 TPF Web-Based Product Support start page

By providing search terms and specifying the information repositories to be selected, you can develop a list of links to Web pages with information related to your search. For example, if you supplied the string **mq** as a search term (see Figure 1) and selected **FAQs** as the search domain, you would get a results page containing links to pages with MQSeries FAQs (see Figure 2).

If you selected **Newsletters** or **TPF Users Group Presentations** as search domains with the same search term of **mq**, you would get a results page containing links to pages with MQSeries information for these categories (not shown). **FAQs** is the default search domain selection.

Note: The highest ranked search results are displayed first.

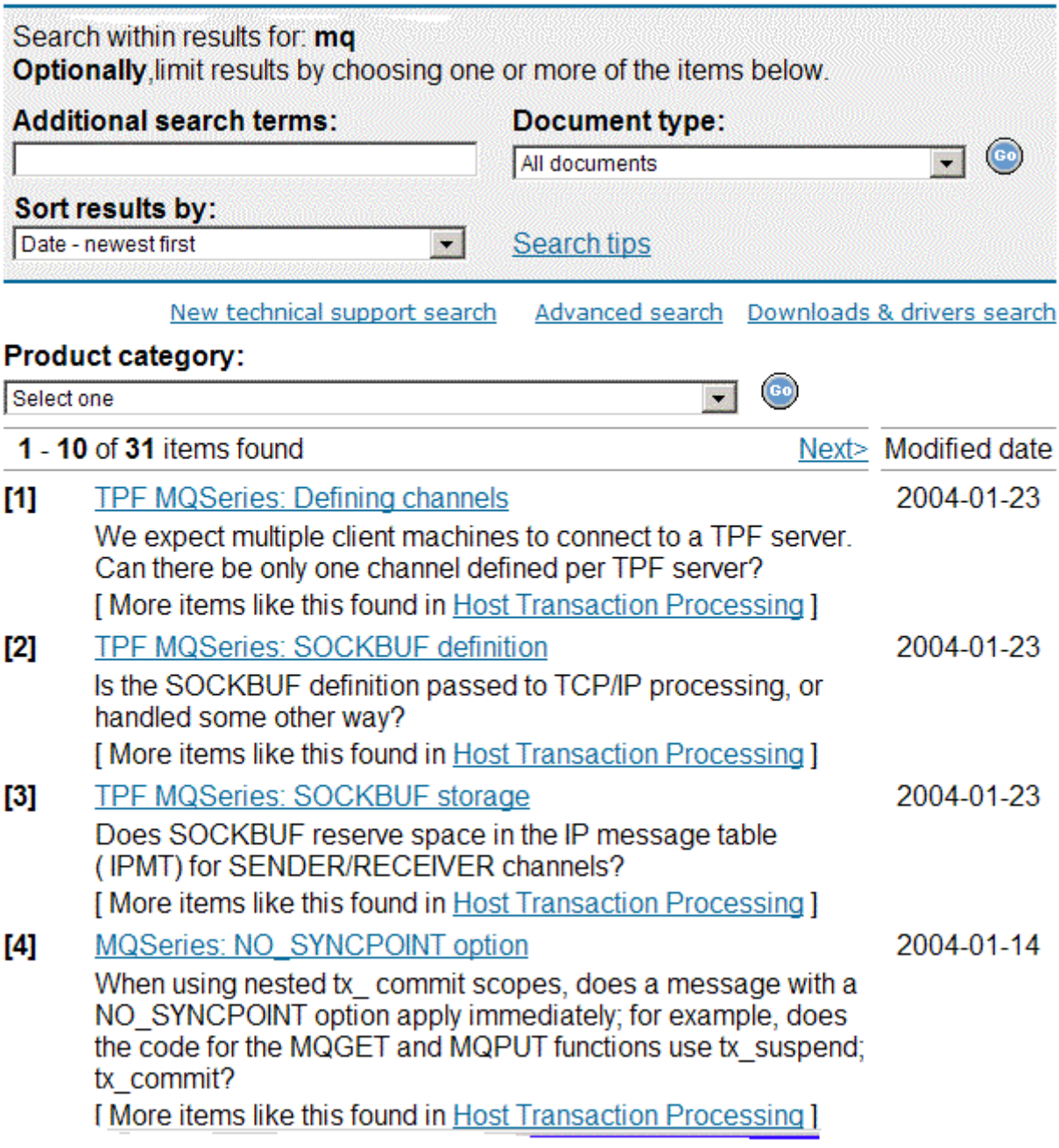
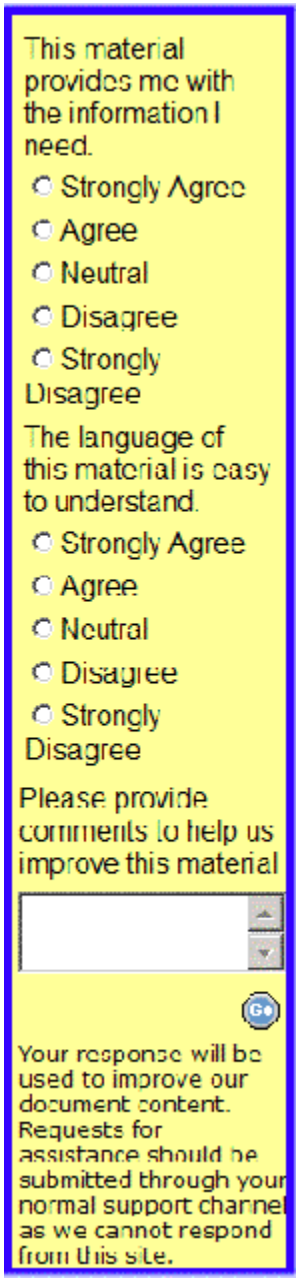


Figure 2 TPF Web-Based Product Support search results

Links

The following links are provided on the right side of the start page (see Figure 1):

- **4.1 Information Center** .
- **TPFDF Information Center**
- **TPF Operations Server**
- **EOCF/2**
- **IIOP® Connect for TPF**
- **How to use this page** .



Feedback

One of the benefits of using TPF Web-Based Product Support is the built-in feedback mechanism, which you can use to rate the information and provide comments. On the right side of the content pages you will see a short survey form similar to that on the right. We provide this survey to determine if the information presented in response to your queries helps to solve your problem and is easy to understand.

The survey gives us valuable feedback and an opportunity to improve the information that we make available to you. Other organizations in IBM use similar surveys to collect and analyze customer feedback.

Navigation

Use the Forward and Back buttons on your browser to return or advance to previously viewed documents or pages. Bookmark the TPF Web-Based Product Support start page to make returning to this area to conduct searches easier.

Searching

Following are some hints and tips for constructing searches on the TPF Web-Based Product Support page.

When composing a Web search, sometimes a little experimentation will help you to get more hits, or a better match between the hits you receive and what you are looking for. If all else fails, try searching for your information on some of the other sites that are linked to TPF Web-Based Product Support.

Documents are indexed for searching using an algorithm called *full text search,* meaning that all significant words in the document (except for words such as *a, and,* and *the*) are displayed in the search index. Documents are ***weighted*** according to how close they come to fulfilling the search, and highly weighted documents are displayed near the top of the list that is returned from the search. For example, if the search argument contained the word Java, a document that had 15 occurrences of Java would be displayed closer to the

top of the list than a document that contained the word Java only once. Other algorithms also are used in weighting to help order the results.

Following are some of the syntax rules for creating a search string:

1. Try to spell out words when possible. An abbreviation from someone else might not be the same as yours. However, if a product or concept has a commonly used acronym, try that first. The word **Include** is better than **inc**, but **PC** is fine for **personal computer**.

2. Use wildcard characters if appropriate. Use a question mark (?) to replace a single character, or use an asterisk (*) to replace one or more characters. For example, **pers*** would return both personal and personality, or **hint?** would return both hint1 and hint2.

3. Case is not significant in a search string. Searching **WebSphere** and **websphere** returns the same results list.

4. You can surround a phrase with quotation marks, but it normally is not necessary. Use quotation marks to narrow down a search. Entering **"TPF system"** or **TPF system** into the search box returns different results.

5. Use logical operators when needed. Use AND to concatenate, OR to provide alternatives, and NOT to negate. However, our experience has shown that NOT has some syntax problems, so you may have to try some alternatives to get the search engine to parse search strings that include NOT. Adding spaces between words and operators sometimes helps.

IBM and the TPF organization have been working to create better ways to locate and retrieve information. Our goal is to provide you with the information that you need when you need it.

Recent TCP/IP Enhancements

Jamie Farmer and Evan Jennings, IBM TPF Development

As the TPF community begins to rely more heavily on TCP/IP, it would help to have better control over IP traffic and the applications themselves. TPF has delivered new TCP/IP enhancements that allow these types of things to happen. Following are some highlights of these new enhancements.

Traffic limiting (APAR PJ28901) allows you to limit the amount of inbound traffic for a specific TPF server application. TPF delivered the connection limiting APAR (PJ28493) on program update tape (PUT) 17. This enhancement allowed you to limit the number of active TCP connections. Although connection limiting allowed you to limit the amount of resources that are used by applications, traffic limiting extends this functionality even further. Traffic limiting works for both TCP and UDP sockets, and allows you to limit inbound traffic for an application, acting as a resource manager for the system. This prevents the system from being flooded by a given application or socket and improves TPF intrusion detection services. Traffic limiting also is used primarily for long-lived connections, while connection limiting works best for short-lived connections.

To limit the amount of inbound traffic for an application, you must define the application in the TPF Network Services Database (NSD). To define an application to the NSD, it must be coded in the /etc/services file. Once the application is defined in the NSD, you can code the *applrate* parameter, *socrate* parameter, or both, for that application. Although both of these parameters limit the amount of inbound traffic for an application, the effect that they have is very different.

The *applrate* parameter specifies the maximum inbound rate in messages per second for all sockets that are used by this application. For example, let's assume that you have a TCP server application and the application currently has five active connections to it. If an *applrate* of 100 was coded for this application, 100 messages per second (at most) would be presented to the application for all five connections combined.

The *socrate* parameter specifies the maximum inbound rate in messages per second for each socket that is used by this application. Let's assume that you have a TCP server application and the application currently has five active connections to it. If a *socrate* of 20 was coded on the application, 20 messages per second (at most) for each active socket connection would be presented to the application.

The *socrate* parameter only can be specified for a TCP server application because UDP is a connectionless protocol; therefore, all remote clients use the same UDP server socket. For TCP sockets, these parameters can be used with each other to further monitor and limit the amount of traffic for a given application.

If a traffic limit is reached in a 1-second time interval, the read API (which includes all read-type APIs) will be delayed until the current time interval expires if running in blocking mode, or will return with the SOCWOULDBLOCK error code if running in nonblocking mode. When the traffic limit is reached, the TPF application is prevented from reading more messages. New messages that are received will be buffered in the socket receive buffer. If the buffer fills up with new messages that are waiting to be read by the application (for TCP), the remote end is prevented from sending more data. For UDP, new input messages will be discarded if the socket receive buffer is full. This is standard UDP behavior and can occur without traffic limiting applied. Implementing traffic limits for an application requires no application code changes because it is the system that slows the rate at which data is presented to the application.

A number of NSD display enhancements are included with the traffic limiting enhancement. For example, you can now display message rates online for a given application. You also can display statistical information about connection limiting and traffic limiting. By using these new displays, you can display the high-water mark for connections or traffic rate and determine how many times a connection or traffic limit was reached for a given application. This becomes very important when tuning your traffic limiting values.

TPF congestion control and avoidance (APAR PJ29144) is another very important APAR that has been provided. With this APAR, the TPF system supports the congestion control algorithm defined in Request for Comments (RFC) 2581. Congestion control is a reactive algorithm that dynamically adjusts the rate at which data is sent to reduce the amount of network congestion and packet loss. The term *reactive algorithm* refers to the fact that once congestion is detected in the network (packets lost, retransmission invoked), the TPF system will slow the rate at which it sends data to the network on a given socket. The congestion control algorithm also incorporates the slow-start algorithm. Slow-start processing is a way to control network congestion by slowly introducing packets to the network when a socket is started (slow-start processing also occurs when there is network congestion).

Along with congestion control, APAR PJ29144 also includes a congestion avoidance algorithm. This is a proactive algorithm that analyzes round-trip times (RTTs) of packets flowing on individual sockets. The term *proactive algorithm* refers to the fact that when the likelihood of congestion is detected, the TPF system will slow the rate at which data is sent to the network. This action is taken before any packets are lost. The combination of congestion control and avoidance results in better end-to-end throughput.

TPF also enhanced the implementations of the congestion control and avoidance algorithms. For example, initial slow-start processing is enabled for all TCP sockets by default. However, for certain applications (such as short-lived connections over local or

high-speed networks), you may not want slow-start processing to be enabled when the socket is started. TPF development created an ioctl() option called TPF_NOSLOWSTART to turn off the initial slow-start processing for a given socket. TPF also enhanced the congestion avoidance algorithm to accommodate sockets with very low RTTs (for example, host-to-host traffic in a data center) and sockets with much higher RTTs (for example, traffic across a wide area network).

With APAR PJ29144 applied to the system, testing showed that end-to-end throughput of a TPF socket was increased significantly. In the comparison test that follows, the time to completion and number of retransmits were tested with and without the congestion control and avoidance APAR applied. The test consisted of 10,000 1400-byte messages being sent from one TPF system as fast as possible, through the network, and back into another TPF system. The following results were obtained:

	Without Congestion Control and Avoidance	**With Congestion Control and Avoidance**
Time to Completion	156 sec	45 sec
Retransmits	5,820	12

The next enhancement is a **TPF API to read a complete TCP message** (APAR PJ29118). The TCP architecture has no concept of a message. Application data usually contains a header in front of the message that contains the length of the message. In cases like this, it is up to the application to parse out the message and involves issuing two or more reads for each message: one or more reads for the length of the message and one or more reads for the message itself. For an AOR, this can result in multiple ECBs being created to read in one message. Not only is this an inefficient process, it has been a common cause of error in application programming.

TPF development has created the following two APIs to alleviate the problems of reading a single TCP message:

tpf_read_TCP_message

activate_on_receipt_of_TCP_message

The format of the message being received is passed to the API. For example, where in the message header does the message length field reside? The system will then get the length of the message, read the entire message, and pass the full message to the application once it is received. Partial messages are never passed to the application.

This enhancement not only makes application programming easier, it is a performance improvement as well. The application issues fewer APIs per message, which increases

the efficiency of the application. This will also cause less dispatching of ECBs and, for AOR, it might reduce the number of ECBs that are created.

For those who have applications that use Systems Network Architecture (SNA) LU 6.2, converting these applications to TCP/IP has just become much easier with the new read message API. LU 6.2 also has structured data messages, meaning that LU 6.2 messages contain a header that contains the length of the message. One of the most difficult tasks in converting LU 6.2 applications to TCP/IP was how the two protocols read data. Now, the LU 6.2 receive_and_wait API can be replaced with the new read message API.

With the addition of these APARs, TPF continues to enhance its TCP/IP stack to better control TCP/IP applications, enhance TCP/IP application design and coding, and increase the performance of the stack itself.

Intruder Alert

Evan Jennings, IBM TPF Development

As the Internet becomes ever more pervasive in our everyday lives, we hear more and more news reports about servers and networks that go down because of malicious activity. Some of the devious attacks unleashed by hackers would make most system administrators sweat. Don't let these reports send you into a panic! TPF gives you a powerful suite of features that help you to detect, avoid, or eliminate the unwanted attention of intruders.

The functions of firewall support (TPF PUT 11 and APAR PJ28213), connection limiting (APAR PJ28493), and traffic limiting (APAR PJ28901) combine to provide Intrusion Detection Services (IDS) and do not require any changes to application code to use them.

Firewall support lets you filter IP packets arriving from the network (Internet or intranet) based on protocol, port or IP address, or any combination thereof, defined by filtering rules in file /etc/iprules.txt. When a packet is rejected or discarded by the TPF firewall as a result of the rules, the TPF IP trace record for the packet includes a reason code. This allows you to identify and log all traffic blocked by the TPF firewall.

Speaking of TPF IP trace, there are over 20 reason codes that can appear in the IP trace for a variety of reasons besides intrusion detection, giving you another useful tool for monitoring network activity and performing problem determination.

Another form of control related to filtering that has existed since PUT 11 is the accept connection user exit (UACC), which allows the TPF installation to verify accept and activate_on_accept requests, and reject them if necessary.

Connection limiting puts a cap on the number of active connections, or sockets, that are allowed at a particular instant for a given TCP application. To use connection limiting, the application must be defined in the Network Services Database (NSD) (defined in file /etc/services) with MAXCONNIN for a TPF server or MAXCONNOUT for a TPF client. If a connection is rejected because the application is already at the connection limit, a reason code is indicated in the TPF IP trace record.

Traffic limiting gives you another tool to throttle traffic to an NSD-defined application either at the application as a whole or at the socket level. Traffic limiting is described in more detail in the "Recent TCP/IP Enhancements" *TPF Systems Technical Newsletter* article.

Connection limiting and traffic limiting prevent a flood of activity on one or even a handful of applications from consuming your system resources at the expense of other critical applications, so they are considered *Quality of Service* (QoS) functions in addition to being part of IDS. The activity that is capped can be the result of a targeted attack or just a natural burst of traffic from your user community.

A type of attack whose purpose is to stop a system from serving its users over the network or otherwise not respond normally is called a *Denial of Service* (DoS) attack. DoS attacks can appear in several forms. One kind is a packet flooding attack, where the recipient is simply overwhelmed with traffic to the point that normal traffic cannot flow. Most types of flood attacks are stopped by filtering, though it would be preferable to have a distributed filtering approach where the edge routers of your network have filtering rules in effect in addition to TPF. This way, flood attacks can be prevented from consuming bandwidth in your network.

Another class of DoS attack works by sending IP packets that are carefully crafted to exploit specific weaknesses known to exist in certain TCP/IP implementations. Some well-known attacks of this type for which TPF is "immunized" against include the Ping of Death (sending a huge ICMP echo message), sending IP fragments with overlapping sequence numbers, SYN attack (starting TCP handshakes, but never completing them), Land Attack (source port and IP address is the same as the destination port and IP address), and TCP RST packets received outside the advertised window. It would be unwise for us to publish a comprehensive list of DoS attacks thwarted by TPF because it would just narrow the focus of a determined hacker. Rest assured that the TPF TCP/IP stack has good

protection against known attacks as well as potential future attacks.

Secure Sockets Layer (SSL) (APAR PJ27863) and Shared SSL support (APAR PJ28118) should also be mentioned. SSL is not part of IDS per se, but SSL provides tools for encrypting data at the socket level. SSL differs from the previously mentioned system functions in that it works through an application programming interface (API), so it must be coded in a C/C++ application in order to use it. What you gain over a nonsecure socket connection is data privacy (no one but the recipient can understand the data), data 0 (24, 29) FDX integrity (any tampering with the data is detected), and authentication (you know without question who you are communicating with). See < "Make Sure Your System Is Securely Fastened While Traffic Is Flowing" in the Third Quarter 2001 issue of the *TPF Systems Technical Newsletter* for more information about SSL.

With the proper use of the powerful tools of IDS, SSL, UACC, and the built-in DoS attack prevention, the TPF system is an extremely uninviting place for intruders.

Intruder alert canceled!

Letters, We Get Letters 10

Bob Kopac, Sarat Vemuri, and P.J. Darcy, IBM TPF Development

This is the 10th article dealing with C and C++ issues. These customer queries usually do not result in PMRs or APARs. Instead, an explanation often clears up customer problems. Sometimes we receive the same question from different customers. Because some of these questions and answers may be of interest to many of you, we decided to write an article based on some of the questions and answers. You might even see one of your questions here! We also include TPF development C and C++ problems so that you can learn from our mistakes. Just don't tell our manager that we make mistakes!

Question: What is the correct way to code to compare a bit field? For example, I have the following definition:

```
struct
   {
                            /* indicators field 1 */
       int isdd_ent_1eot :1; /* end of table indicator */
       int isdd_ent_selt :1; /* selective tracing active */
       int               :6; /* spares */
    }  isdd_ent_ind1;
```

I want to test one of the bits; for example:

```
    if (sdd_ent_ptr->isdd_ent_ind1.isdd_ent_1eot == 1)
```

Is this correct?

Answer: No, it is incorrect. Remember that every bit counts. A 1-bit field needs to be defined as an *unsigned int* type. The minimum width of a *signed int* bit field is 2 bits. You

should change your structure to use the *unsigned int* type instead of the *int* type.

Question: Why does my compiler listing show all C++ statements grouped together followed by all compiler-generated assembler instructions? Usually I see the C++ statements interleaved with the compiler-generated assembler instructions.

Answer: You used the **NOTEST(NOHOOK)** compiler option. This option groups the C++ statements together. Coding **NOTEST(HOOK)** makes it easier to match the C++ statements with the assembler code; for example:

```
                    00165 | *  sipcc_parms.sipcc_da2_len = 0;
0002F6  5060  D0F0  00165 |        ST    r6,<a1:d240:l4>(,r13,240)
                    00166 | *  sipcc_parms.sipcc_priority = SIPCC_PRIORITY_YE

0002FA  50E0  D0F8  00166 |        ST    r14,<a1:d248:l4>(,r13,248)
```

Question: I have a question about double-byte character set (DBCS) support in TPF. Are wide character support and multibyte character support the same as DBCS?

Answer: Yes, they are. The *TPF 4.1 Glossary* has the following definitions:

> **DBCS** Double-byte character set.
> **SBCS** Single-byte character set.
> **multibyte characters** A mixture of SBCS and DBCS characters.

TPF 4.1 APAR PJ21337 implemented the OS /390 DBCS double-byte wide character implementation and the OS/390 multibyte implementation. APAR PJ21337 does not support any other non-OS/390 wide character implementation. Note that there are other non-IBM wide character representations that may have different values for a character and may represent the character using more than 2 bytes.

For reading or writing multibyte characters, APAR PJ21337 updated the existing ISO-C printf() and scanf() functions to support new format characters for multibyte characters.

APAR PJ21337 added new functions on TPF to manipulate multibyte and wide character data; for example:

function	Description
swprintf()	Write wide characters to a wide-character array.
swscanf()	Read a wide-character string.
wcstombs()	Convert a wide-character string to a multibyte string.
mbstowcs()	Convert a multibyte string to a wide-character string.

Appendix D ("C/C++ Functions Supported by the TPF 4.1 System but Not Documented") in the *TPF C/C++ Language Support User's Guide* lists these new functions, which are documented in the *OS/390 C/C++ Run-Time Library Reference* document. Following is a link:

http://www.ibm.com/software/awdtools/c390/cmvsdocs.htm

See "APAR PJ21337 — A Wide Character Is Not Godzilla" in the Second Quarter 2000 issue of the *TPF Systems Technical Newsletter* to help you gain a better understanding of the difference between wide characters (DBCS) and multibyte characters.

Note: You should use multibyte formatting, especially when writing data to DASD. You may use wide characters for processing strings in core.

Note: Do not write wide character data to DASD.

Question: Is it possible for C++ to "ENTRC" to an E-type TPF assembler program?

Answer: Yes, it is possible. You need to create a function prototype for the call to the assembler segment and surround the prototype with *extern "C" { }* . This makes the call interface a C function interface rather than a C++ function interface. The function name is then resolved with a DLM stub, which provides linkage to enter/back services.

Following is an example of a C++ program (qzz3) calling an assembler segment (QZZ1):

```
    #include <stdlib.h>
    #include <stdio.h>
    #include <tpfregs.h>

    extern "C" { void QZZ1 (struct TPF_regs *); } /* function prototyp
e
                                                    for assembler segment */
    void qzz3 (void)
    {
    struct TPF_regs * regs;
    QZZ1(regs);
    exit(0);
    }
```

Follow-Up Question: Could you explain further why I have to code *extern "C"* ?

Answer: The interface to the assembler program uses C linkage and not C++ linkage; for C linkage, register 1 points to a parameter list. The interface does not look like a C++ interface. That is why you have to wrap the prototype with the extern "C" wrapper. See "Grandson of Letters, We Get Letters" in the Second Quarter 2001 *TPF Systems Technical Newsletter* for a question and answer about extern "C". The article includes a paragraph that explains the difference between the C interface and the C++ interface.

Follow-Up Question: Can an assembler program call a DLL?

Answer: No.

Follow-Up Question: Can you CRETC to a DLL?

Answer: No.

Follow-Up Question: Is it possible to do a CROSC in assembler and pass parameters to a C DLM? If I set register 1 to point to my parameter list and then do a CROSC, will my parameters show up in the DLM arguments as they do on an ENTRC?

Answer: Yes, it can be done that way for a C program. It would not work for a C++ program unless the C++ prototype is wrapped in an extern "C" wrapper so that the parameters passed and received are in a C format, not a C++ format.

Question: Why am I receiving compiler error message CBC1055(S)? Following is an example of the error message.

```
    imqt_ent_struct* curr_mqt_ptr;
    ...
    curr_mqt_ptr = malloc (MQIT_MAX_ENTRIES * IMQT_ENT_LEN);
    CBC1055(S) "void*" cannot be converted
 to "imqt_ent_struct*"
```

Answer: The stdlib.h header contains the following prototype for malloc:

```
    void * malloc (size_t);
```

That is, malloc() returns a pointer to void. The compiler expects the *lvalue* to be a pointer to *void* type. You coded *curr_mqt_ptr* to be a pointer to *imqt_ent_struct* and not as a pointer to *void*. The compiler detects that a pointer to *void* is not the same as a pointer to *imqt_ent_struct* and indicates this with the CBC1055(S) message. This error message may seem annoying, but remember that programmers sometimes erroneously code a pointer to the wrong structure, and this compiler error message finds those problems.

You should do ***one*** of the following.

- Code the *lvalue* (the value to the left of the malloc() statement) to be a pointer to *void*.
- Cast the return value of the function to be a pointer as follows:

```
curr_mqt_ptr =
      (imqt_ent_struct*) malloc (MQIT_MAX_ENTRIES * IMQT_ENT_LEN);
```

Optionally, compiling with the NOANSIALIAS compiler option may get rid of the problem.

Note: You should always compile TPF-provided code using the NOANSIALIAS compiler option. The ANSIALIAS/NOANSIALIAS compiler option default is ANSIALIAS. This default assumes that pointers access objects of the same type; that is, the pointers are ANSI compliant. Casting a pointer to point to a different object is a common C practice, but it is not ANSI compliant. If your code casts a pointer to point to a different object, you must then compile your code using the NOANSIALIAS compiler option. See "Letters, We Get Letters . . . Usually C and C++" in the Fourth Quarter 2000 *TPF Systems Technical Newsletter* for additional questions and answers about NOANSIALIAS.

Question: Why are QCON references relocated to the static block for each ECB? Function calls within the same .cpp file are resolved with an ADCON. Function calls across .cpp files in the same DLM or DLL (not calls to exported functions) are resolved with a QCON instead of a VCON. Merging two .cpp files into one .cpp file eliminates this overhead. I wonder why these are not resolved with a VCON?

Answer: While compiling a .cpp source unit, the compiler does not know the difference between calls to functions in a different source unit versus functions in a different module. For C++ or DLL applications, calls to any external function are via DLL linkage, which uses the descriptor that resides in static storage. Thus, for external function calls (external to the source unit being compiled), the compiler generates a QCON reference.

Question: If I have a DLM that does not have a main() function but has a 4-character TPF entry point function, and the function uses static, will its static block be preserved between subsequent calls to that DLM, or will its static block be reallocated and reinitialized on every call to that DLM?

Answer: The static block is reused for the DLM for the life of the ECB. Each ECB gets its own static block for that DLM. Thus, static blocks exist for the life of the ECB and exist across multiple invocations.

The static frame is used for each C DLM that uses reentrant static storage. A Hash table is used to locate the static frame associated with each C program. Static storage frames are allocated and initialized using descriptors produced by the compiler. Extern variables are the same as static variables with the exception that extern variables can be accessed by all functions within a DLM.

The rules for function, file, and module scope are as follows: 0 (24, 29) FDX Every definition that has a *static* qualifier becomes file scope. If that definition (with a *static* qualifier) is within a function, it becomes function scope. Any definition that is outside of a function without the *static* qualifier becomes externalized to module scope.

Question: Are those all the questions you received from customers and TPF developers?

Answer: No, there have been many questions. We have enough material for several newsletter articles. We hope that is a good thing!

IBM Corporation
TPF Systems Development
Mail Station P923
2455 South Road
Poughkeepsie, NY 12601-5400
USA